

### Preparing environment

<code>mkdir project_name &amp;&amp; cd \$_</code>	Create project folder and navigate to it
<code>python -m venv env_name</code>	Create venv for the project
<code>source env_name \bin \activate</code>	Activate environment (Replace "bin" by "Scripts" in Windows)
<code>pip install django</code>	Install Django (and others dependencies if needed)
<code>pip freeze &gt; requirements.txt</code>	Create requirements file
<code>pip install -r requirements.txt</code>	Install all required files based on your pip freeze command
<code>git init</code>	Version control initialisation, be sure to create appropriate <a href="#">gitignore</a>

### Create project

<code>django-admin startproject mysite (or I like to call it config)</code>	This will create a mysite directory in your current directory the <code>manage.py</code> file
<code>python manage.py runserver</code>	You can check that everything went fine

### Database Setup

<code>Open up mysite/settings.py</code>	It's a normal Python module with module-level variables representing Django settings.
<code>ENGINE = 'django.db.backends.sqlite3' or 'django.db.backends.postgresql', 'django.db.backends.mysql', or 'django.db.backends.oracle'</code>	If you wish to use another database, install the appropriate database <a href="#">bindings</a> and change the following keys in the DATABASES 'default' item to match your database connection settings
<code>NAME</code> – The name of your database. If you're using SQLite, the database will be a file on your computer; in that case, NAME should be the full absolute path, including filename, of that file.	The default value, <code>BASE_DIR / 'db.sqlite3'</code> , will store the file in your project directory.
If you are not using SQLite as your database, additional settings such as <code>USER</code> , <code>PASSWORD</code> , and <code>HOST</code> must be added.	For more details, see the reference documentation for <a href="#">DATABASES</a> .

### Creating an app

<code>python manage.py startapp app_name</code>	Create an <code>app_name</code> directory and all default file/folder inside
<code>INSTALLED_APPS = [     'app_name',     ...</code>	Apps are "pluggable", that will "plug in" the app into the project



By **Olivier R. (OGR)**  
[cheatography.com/ogr/](https://cheatography.com/ogr/)

Published 6th February, 2022.  
Last updated 12th February, 2022.  
Page 1 of 8.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!  
<https://apollopad.com>

### Creating an app (cont)

```
urlpatterns = [
    path('app_name/', include('app_name.urls')),
    path('admin/', admin.site.urls),
]
```

Into urls.py from project folder, include app urls to project

### Creating models

```
class ModelName(models.Model):
    # ...
```

Create your class in the app\_name/models.py file

```
title = models.CharField(
    max_length=100)
```

Create your fields

```
def __str__(self):
    return self.title
```

It's important to add \_\_str\_\_() methods to your models, because objects' representations are used throughout Django's automatically-generated admin.

### Database editing

```
python manage.py makemigrations
(app_name)
```

By running makemigrations, you're telling Django that you've made some changes to your models

```
python manage.py sqlmigrate #identifier
```

See what SQL that migration would run.

```
python manage.py check
```

This checks for any problems in your project without making migrations

```
python manage.py migrate
```

Create those model tables in your database

```
python manage.py shell
```

Hop into the interactive Python shell and play around with the free API Django gives you

### Administration

```
python manage.py create_superuser
```

Create a user who can login to the admin site

```
admin.site.register(ModelName)
```

Into app\_name/admin.py, add the model to administration site

```
http://127.0.0.1:8000/admin/
```

Open a web browser and go to "/admin/" on your local domain

### Management

```
mkdir app_name/management app_name/management/commands &&
cd $_
```

Create required folders

```
touch your_command_name.py
```

Create a python file with your command name

```
from django.core.management.base import BaseCommand
#import anything else you need to work with (models?)
```

Edit your new python file, start with import



### Management (cont)

```
class Command(BaseCommand):
    help = "This message will be shown with the --help option after
your command"
```

Create the Command class that will handle your command

```
def handle(self, args, *kwargs):
    # Work the command is supposed to do
```

```
python manage.py my_custom_command
```

And this is how you execute your custom command

Django lets you create your custom CLI commands

### Write your first view

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world. You're at the
index.")
```

Open the file `app_name/views.py` and put the following Python code in it.

This is the simplest view possible.

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

In the `app_name/urls.py` file include the following code.

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

### View with argument

```
def detail(request, question_id):
    return HttpResponse(f"You're looking at question {question_id}")
```

Example of view with an argument

```
urlpatterns = [
    path('<int:question_id>/', views.detail, name='detail'),
    ...
```

See how we pass argument in path

```
{% url 'app_name:view_name' question_id %}
```

We can pass attribute from template this way

### View with Template

```
app_name/ templates /app_name /index.html
```

This is the folder path to follow for template

```
context = {'key': value}
```

Pass values from view to template

```
return render(request, 'app_name /index.html',
context)
```

Example of use of render shortcut

```
{% Code %}
```

Edit template with those. Full list [here](#)

```
{{ Variable from view's context dict }}
```

```
<a href="{% url 'detail' question_id %}" ></a>
```

```
<title >Page Title< /title>
```

you can put this on top of your html template to define page title



### Add some static files

<code>'django.contrib.staticfiles'</code>	Be sure to have this in your INSTALLED_APPS
<code>STATIC_URL = 'static/'</code>	The given examples are for this config
<code>mkdir app_name/ static app_name/ static /app_name</code>	Create static folder associated with your app
<code>{% load static %}</code>	Put this on top of your template
<code>&lt;link rel="stylesheet" type="text/css" href="{% static 'app_name/stylesheet.css' %}"&gt;</code>	Example of use of static.

### Raising 404

<code>raise Http404("Question does not exist")</code>	in a try / except statement
<code>question = get_object_or_404(Question, pk=question_id)</code>	A shortcut

### Forms

<code>app_name/forms.py</code>	Create your form classes here
<code>from django import forms</code>	Import django's forms module
<code>from .models import YourModel</code>	import models you need to work with
<code>class ExampleForm(forms.Form):</code> <code>example_field = forms.CharField(label='Example label', max_length=100)</code>	For very simple forms, we can use simple Form class
<code>class ExampleForm(forms.ModelForm):</code> <code>class meta:</code> <code>model = model_name</code> <code>fields = ["fields"]</code> <code>labels = {"text": "label_text"}</code> <code>widget = {"text": forms.Widget_name}</code>	A ModelForm maps a model class's fields to HTML form <input> elements via a Form. Widget is optional. Use it to override default widget
<code>TextInput, EmailInput, PasswordInput, DateInput, Textarea</code>	Most common widget <a href="#">list</a>
<code>if request.method != "POST":</code> <code>form = ExampleForm()</code>	Create a blank form if no data submitted
<code>form = ExampleForm(data=request.POST)</code>	The form object contains the information submitted by the user
<code>if form.is_valid():</code> <code>form.save()</code> <code>return redirect("app_name:view_name", arguments=arguments)</code>	Form validation. Always use redirect function
<code>{% csrf_token %}</code>	Template tag to prevent "cross-site request forgery" attack



### Render Form In Template

<code>{{ form.as_p }}</code>	The most simple way to render the form, but usually it's ugly
<code>{{ field placeholder:"field label" }}</code> <code>{{ form.username placeholder:"Your name here" }}</code>	The <code> </code> is a filter, and here for placeholder, it's a custom one. See next section to see how to create it
<code>{% for field in form %}</code> <code>{{form.username}}</code>	You can extract each field with a for loop. Or by explicitly specifying the field

### Custom template tags and filters

<code>app_name \template_tags \__init__.py</code>	Create this folder and this file. Leave it blank
<code>app_name \template_tags \filter_name.py</code>	Create a python file with the name of the filter
<code>{% load filter_name %}</code>	Add this on top of your template
<code>from django import template</code>  <code>register = template.Library()</code>	To be a valid tag library, the module must contain a module-level variable named <code>register</code> that is a <code>template.Library</code> instance
<code>@register.filter(name='cut')</code> <code>def cut(value, arg):</code> <code>""" Removes all values of arg from the given string """</code> <code>return value.replace(arg, '')</code>	Here is an example of filter definition. See the decorator? It registers your filter with your Library instance. You need to restart server for this to take effects
<a href="https://tech.serhatteker.com/post/2021-06/placeholder-template-tags/">https://tech.serhatteker.com/post/2021-06/placeholder-template-tags/</a>	Here is a link of how to make a placeholder custom template tag

### Setting Up User Accounts

Create a "users" app	Don't forget to add app to settings.py and include the URLs from users.
<code>app_name = "users"</code> <code>urlpatterns[</code> <code># include default auth urls.</code> <code>path("", include("django.contrib.auth.urls"))</code> <code>]</code>	Inside <code>app_name/urls.py</code> (create it if inexistent), this code includes some default authentication URLs that Django has defined.
<code>{% if form.error %}</code> <code>&lt;p&gt;Your username and password didn't match&lt;/p&gt;</code> <code>{% endif %}</code> <code>&lt;form method="post" action="{% url 'users :login' %}"&gt;</code> <code>{% csrf_token %}</code> <code>{{ form.as_p }}</code>  <code>&lt;button name="submit"&gt;Log in&lt;/button&gt;</code> <code>&lt;input type="hidden" name="next" value="{% url 'app_name :index' %}" /&gt;</code> <code>&lt;/form&gt;</code>	Basic login.html template Save it as save template as <code>users/templates/registration/login.html</code> We can access to it by using <code>&lt;a href="{% url 'users :login' %}"&gt;Log in&lt;/a&gt;</code>



### Setting Up User Accounts (cont)

<code>{% if user.is_authenticated %}</code>	Check if user is logged in
<code>{% url "users:logout" %}</code>	Link to logout page, and log out the user save template as <code>users/templates/registration/logged_out.html</code>
<code>path("register/", views.register, name="register")</code> ,	Inside <code>app_name/urls.py</code> , add path to register
<pre>from django.shortcuts import render, redirect from django.contrib.auth import login from django.contrib.forms import UserCreationForm  def register(request):     if request.method != "POST":         form = UserCreationForm()     else:         form = UserCreationForm(data=request.POST)      if form.is_valid():         new_user = form.save()         login(request, new_user)         return redirect("app_name:index")  context = {"form": form} return render(request, "registration/register.html", context)</pre>	We write our own <code>register()</code> view inside <code>users/views.py</code> For that we use <code>UserCreationForm</code> , a Django building model. If method is not post, we render a blank form Else, is the form pass the validity check, an user is created We just have to create a <code>registration.html</code> template in same folder as the <code>login</code> and <code>logged_out</code>

### Allow Users to Own Their Data

<code>...</code>	Restrict access with <code>@login_required</code> decorator
<pre>from django.contrib.auth.decorators import login_required ...</pre>	If user is not logged in, they will be redirect to the login page To make this work, you need to modify <code>settings.py</code> so Django knows where to find the login page
<pre>@login_required def my_view(request)     ...</pre>	Add the following at the very end <code># My settings</code> <code>LOGIN_URL = "users:login"</code>
<code>...</code>	Add this field to your models to connect data to certain users
<pre>from django.contrib.auth.models import User ...</pre>	When migrating, you will be prompt to select a default value
<pre>owner = models.ForeignKey(User, on_delete= models.CASCADE)</pre>	



### Allow Users to Own Their Data (cont)

```
user_data = ExampleModel.objects.filter(owner = request.user)
```

```
...
```

```
from django.http import Http404
```

```
...
```

```
...
```

```
if example_data.owner != request.user:
    raise Http404
```

```
new_data = form.save(commit=False)
```

```
new_data.owner = request.user
```

```
new_data.save()
```

Use this kind of code in your views to filter data of a specific user

request.user only exist when user is logged in

Make sure the data belongs to the current user

If not the case, we raise a 404

Don't forget to associate user to your data in corresponding views

The "commit=False" attribute let us do that

### Paginator

```
from django.core.paginator import Paginator
```

In app\_name/views.py, import Paginator

```
example_list = Example.objects.all()
```

In your class view, Get a list of data

```
paginator = Paginator(example_list, 5) # Show 5 items per page.
```

Set appropriate pagination

```
page_number = request.GET.get('page')
```

Get actual page number

```
page_obj = paginator.get_page(page_number)
```

Create your Page Object, and put it in the context

```
{% for item in page_obj %}
```

The Page Object acts now like your list of data

```
<div class="pagination">
```

An exemple of what to put on the bottom of your page to navigate through Page Objects

```
<span class="step-links">
```

```
{% if page_obj.has_previous %}
```

```
<a href="?page=1" >&laquo; first</a>
```

```
<a href="?page={{ page_obj.previous_page_number
```

```
}}">previous</a>
```

```
{% endif %}
```

```
<span class="current"> Page {{ page_obj.number }} of {{ page_obj.paginator.num_pages }}. </span>
```

```
{% if page_obj.has_next %}
```

```
<a href="?page={{ page_obj.next_page_number }}">next</a>
```

```
<a href="?page={{ page_obj.paginator.num_pages }}">last &rarr;</a>
```

```
</div>
```

```
{% endif %}
```

```
</span>
```

```
</div>
```



### Deploy to Heroku

<https://heroku.com>

Make a Heroku account

<https://devcenter.heroku.com/articles/heroku-cli/>

Install Heroku CLI

```
pip install psychog2
pip install django -heroku
pip install gunicorn
```

install these packages

```
pip freeze > requir em e n ts.txt
```

update requirements.txt

```
# Heroku settings.
import django _heroku
django _he rok u.s ett ing s(l oca ls(),
static fil es= False)
if os.env iro n.g et( 'DE BUG') == " TRU E":
    DEBUG = True
elif os.env iro n.g et( 'DE BUG') == " FAL -
SE":
    DEBUG = False
```

At the very end of settings.py, make an Heroku ettings section import django \_heroku and tell django to apply django heroku settings  
The staticfiles to false is not a viable option in production, check whitenoise for that IMO

C

By **Olivier R. (OGR)**  
[cheatography.com/ogr/](https://cheatography.com/ogr/)

Published 6th February, 2022.  
Last updated 12th February, 2022.  
Page 8 of 8.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish  
Yours!  
<https://apollopad.com>