

**Eric The Coder**

Posted on Apr 5 • Updated on Sep 15

My beloved Django cheat sheet

#python #django #beginners

Follow me!: [Follow @EricTheCoder](#)

Start a new Django project

Create et access project folder

~\$ mkdir project_name

~\$ cd project_name

Create Python virtual env

~\$ python3 -m venv venv

Activate virtual env

~\$ source venv/bin/activate

If you want to deactivate virtual env

~\$ deactivate

```
# Install django (~= same as 3.1.*)
~$ pip install django~=3.1.0

# New django project (from project_name folder)
~$ django-admin startproject config .

# Create app (from project_name folder)
~$ python manage.py startapp app_name
```

Migration:

Django create a database table for each models present in your app using thoses commands:

- **Makemigrations:** Create a file under app_name/migrations with the database structure to create

```
~$ python manage.py makemigrations
```

- **Migrate:** Will read the migrations files and create the actual database and tables

```
~$ python manage.py migrate
```

Create superuser for authentication/admin panel

```
~$ python manage.py createsuperuser
```

Start server

```
~$ python manage.py runserver => ex. http://127.0.0.1:8000
```

Requirements

```
# Create a requirements file that contain all your projet dependencies
~$ pip freeze > requirements.txt

# Install your project requirements (if a requirements file exist)
~$ pip install -r requirements.txt
```

Other commands

```
# Django shell (Run projet code direclty)
~$ python manage.py shell
```

```
# example of code to run in the shell:
>>> from app_name.models import User
>>> user1 = User.objects.first()

# Prepare static folders for production
$ python manage.py collectstatic

# Take all data from app blog and export in json
python manage.py dumpdata blog >myapp.json

# Take all data in json file and import in app data table
python manage.py loaddata myapp.json
```

Project config

```
# Add app to settings.py
INSTALLED_APPS = [ ..., 'app_name' ]

# App templates folder
create folder appfolder/templates/appname

# Project templates folder:
create folder projectname/templates

# settings.py template config
Project templates settings.py:
    TEMPLATES = [
        { ...
            'DIRS': [BASE_DIR / 'templates', ],
        ... }

# Create Static folder:
project_name\static\

# Static folder (settings.py):
STATIC_URL = '/static/'
STATICFILES_DIRS = [ BASE_DIR / 'static' ]
STATIC_ROOT = 'static_root'

# To use PostgreSQL
# pip install psycopg2
# settings.py
DATABASE = {
```

```
'default': {
    'ENGINE': 'django.db.backends.postgresql',
    'NAME': 'blog',
    'USER': 'admin',
    'PASSWORD': '123456',
    'HOST': 'localhost',
    'PORT': '5432'
```

Create data model:

Theses models can be created as database tables with the migrations commands

```
# models.py
# The id fields is automaticly created by Django for each model that why it's not show be

from django.db import models

class Customer(models.Model)
    name = models.CharField('Customer', max_length=120)
    age = models.IntegerField()
    note = models.TextField(blank=True, null = True)
    email = models.EmailField(max_length=255, blank=True, null=True)
    credit = models.FloatField(blank=True)
    is_active = models.BooleanField(default=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # Select Field (return value, display value)
    TYPE_CHOICES = (
        ('Customer', 'Customer'),
        ('Supplier', 'Supplier'),
        ('Student', 'Student'),
    )

    type = models.CharField(choices=TYPE_CHOICES)
```

Model string representation

```
class Customer(models.Model):
    name = models.CharField('Customer', max_length=120)
    age = models.IntegerField()

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

Relationship between models

```
# One-to-Many: (use double quotes if the entity is not yet declare) ex. "Supplier"
supplier = models.ForeignKey(Supplier, blank=True, null=True, on_delete=models.CASCADE)

# on_delete can be set to models.CASCADE, models.ST_DEFAULT or models.SET_NULL

# Many-to-Many:
tags = models.ManyToManyField(Tag, blank=True)

# One to One
User = models.OneToOneField(User, on_delete=models.CASCADE)

# Overwrite save method
def save(self, *args, **kwargs):
    if not self.slug:
        self.slug = slugify(self.title)

    super().save(*args, **kwargs)
```

Admin panel:

Every Django projects come with an Admin Panel that can be open at /admin url (ex: localhost:8000/admin)

To display the model in the Admin panel register the model in the app_name/admin.py file

```
from .models import Blog
admin.site.register(Blog)
```

Customize Admin Panel

For each models you can specify the fields you want to use

```
# Custom model Admin (admin.py):
class BlogAdmin(admin.ModelAdmin)
    fields = ("title", "description") # Fields to use for add/edit/show page
    list_display = ("title", "description") # fields to display in search page
    list_display_links = ("title") # fields that will be a link in search page
    ordering = ("date_created",) # Ordering allowed in the search page
    search_fields = ("title", "description") # Search fields allowed in the search page

# Register app
admin.site.register(Blog, BlogAdmin)
```

Routing:

Django routing info is store in project_folder/urls.py file

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

urlpatterns = [
    path('admin/', admin.site.urls), # pre-created admin urls routes
    path('', include('app_name.urls')) # include your app urls
]
```

the 'include()' method allow to link another urls.py file created in your app folder (app_name/urls.py)

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

url patterns = [
    path('posts', views.index, name='posts.index'),
    path('posts/create/', views.create, name='posts.create'),
    path('posts/<int:id>', views.show, name='posts.show'),
    path('posts/<int:id>/edit/', views.edit, name='posts.edit'),
    path('posts/<int:id>/delete/', views.delete, name='posts.delete'),
]
```

Static route

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Function Based Views

```
# views.py
from django.shortcuts import render, redirect
from .models import Post
from .forms import PostForm

def index(request):
    # Get all Posts
```

```
posts = Post.objects.all()

# Render app template with context
return render(request, 'appfolder/index.html', {'posts': posts})

def show(request, id):
    post = Post.objects.get(id=id)
    return render(request, 'appfolder/show.html', {'post': post})

def create(request):
    form = PostForm(request.POST or None)
    if form.is_valid():
        # optionally we can access form data with form.cleaned_data['first_name']
        post = form.save(commit=False)
        post.user = request.user
        post.save()
        return redirect('/posts')

    return render(request, 'appfolder/create.html', {'form': form})

def edit(request, id):
    post = Post.objects.get(id=id)
    form = PostForm(request.POST or None, instance=post)
    if form.is_valid():
        form.save()
        return redirect('/posts')

    return render(request, 'appfolder/edit.html', {'form': form})

def delete(request, id):
    post = Post.objects.get(id=id)
    post.delete()
    return redirect('/posts')
```

Class based Views

```
from django.views.generic import TemplateView, ListView, DetailView, CreateView, UpdateView

class LandingPageView(TemplateView):
    template_name = 'landing.html'

    # Optional: Change context data dict
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
```

```
        context['title'] = 'Landing Page'
    return context

class PostsListView(ListView):
    queryset = Post.objects.all()

    # Optional
    # context_object_name = "posts" (default: post_list)
    # template_name = 'posts.html' (default: posts/post_list.html)

class PostsDetailView(DetailView):
    model = Post # object var in template

    # Optional
    # template_name = 'post.html' (default: posts/post_detail.html)

class PostsCreateView(CreateView):
    form_class = PostForm

    template_name = 'posts/post_create.html' # no default value

    def get_success_url(self):
        return reverse('posts-list')

    # Optional: Overwrite form data (before save)
    def form_valid(self, form):
        if self.request.user.is_authenticated:
            form.instance.author = self.request.user

        return super().form_valid(form)

class PostsUpdateView(UpdateView):
    model = Post
    form_class = PostForm
    template_name = 'posts/post_update.html'

    def get_success_url(self):
        return reverse('post-list')

    # Optional: Change context data dict
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['submit_text'] = 'Update'
        return context
```



```
class PostsDeleteView(DeleteView):
    model = Post
    template_name = 'posts/post_delete.html'
    success_url = reverse_lazy('posts-list')

# Urls.py route declaration
path('<int:pk>/update/', PostsUpdateView.as_view(), name='post-update')
```

Django Template:

Templates are store in project_folder/templates or in your app_folder/templates/app_name/*.html

```
{% extends 'base.html' %}
{% block content %}
{% endblock %}

{% include 'header.html' %}

{% if user.username = 'Mike' %}
    <p>Hello Admin</p>
{% else %}
    <p>Hello User</p>
{% endif %}

{% for product in products %}
    <p>The product name is {{ product }}<p>
{% endfor %}

{{ var_name }}
```

Template variables formating

```
{{ title | lower }}
{{ blog.post | truncatwords:50 }}
{{ order.date | date:"D M Y" }}
{{ list_items | slice:"3" }}
{{ total | default:"nil" }}
```

Current path (ex. posts/1/show)

```
{{ request.path }}
```

URL by name with param

```
{% url 'posts.delete' id=post.id %}
```

Use static in template:

```
{% load static %}
{% static 'css/main.css' %}
```

Model Managers and Querysets:

Model manager allow model database reads and writes

One line create and save

```
Article.objects.create(name='Item 1', price=19.95)
```

Read all

```
Article.objects.all()
```

Create

```
user = User.objects.first()
```

```
article = Article(user=user, name='Item 1', price=19.95)
```

Save

```
article.save()
```

Read one

```
Article.objects.get(id=1)
```

Select Related (to avoid n+1 query)

```
posts = Post.objects.select_related('user', 'category').all()
```

Read or render a 404 not found page

```
from django.shortcuts import get_object_or_404
```

```
article = get_object_or_404(Article, id=512)
```

Filter

```
Article.objects.filter(model='dyson', name__icontains='dyson') # __icontains
```

```
Article.objects.filter(year__gt=2016) # __gt = greater than
```

```
Article.objects.filter(year__lt=2001) # __lt = less than
```

Filter on relationship sub model field

```
Article.objects.get(user__username='mike')
```

Ordering

```
Article.objects.order_by('name') # ascending
```

```
Article.objects.order_by('-name') # descending
```

Slicing return first

```
Article.objects.all().order_by('name')[0]

# Slicing return last
Article.objects.all().order_by('-name')[0]

# Slicing limit/offset
Article.objects.all().order_by('name')[1..10]

# Updating
article = Article.objects.first()
article.name = 'new name'
article.save()

# One line update
Article.objects.filter(id=4).update(name='new name')

# Deleting
article = Article.objects.first()
article.delete()

# One line delete
article.objects.get(id=1).delete()

# Delete all
Article.objects.all().delete()

# Set ForeignKey field value
model1 = Model(name='dyson')
article.model = model1

# Get ForeignKey value
article1.model.name
model1.article_set.all()

# Add Many-to-Many
article1.tags.add(tag1)
article1.tags.all()
tag1.articles_set.all()
```

Form (forms.py)

```
from django import forms
class ArticleForm(forms.Form):
    name = forms.CharField(max_length=100)
```

```
description = forms.CharField(blank=True, null = True)
```

```
# Model Form
```

```
from django.forms import ModelForm
from .models import Article
class ArticleForm(ModelForm):
    class Meta:
        model = Article
        fields = ['name', 'description', 'price'] # Use '__all__' for all fields
```

```
# Render form in template
```

```
<form method="post" action="" novalidate>
    {% csrf_token %}
    {{ form }}
    <button type="submit">Submit</button>
</form>
```

```
# Bootstrap (pip install django-crispy-forms + installed_apps: 'crispy_forms')
```

```
{% load crispy_forms_tags %}
{{ form|crispy }}
{{ form.email|as_crispy_field }}
```

```
# crispy-tailwind
```

```
pip install crispy-tailwind
```

```
# settings.py
```

```
CRISPY_ALLOWED_TEMPLATE_PACKS = 'tailwind'
CRISPY_TEMPLATE_PACK = 'tailwind'
```

```
# template usage
```

```
{% load tailwind_filters %}
{{ form|crispy }}
```

Form validation

```
# forms.py
```

```
from django.core.exceptions import ValidationError
```

```
# field validation
```

```
def clean_first_name(self):
    data = self.cleaned_data['first_name']
    if data == 'Mike':
```

```

        raise ValidationError('Your name must not be Mike')

    return data

# form validation
def clean(self):
    first_name = self.cleaned_data['first_name']
    last_name = self.cleaned_data['last_name']
    if first_name + last_name == 'MikeTaylor':
        raise ValidationError('Your name must not be Mike Taylor')

```

Flash messages

```

messages.success(request, 'Login successful')
messages.error(request, 'Login error')

```

```

# Message tags
# debug, info, success, warning and error

```

```

# Display flash in template
{% if messages %}
    {% for message in messages %}
        {% message %}
        {% message.tags %}
    {% endfor %}
{% endif %}

```

User model (pre-created)

```

# Get a reference to Django pre-created User model
from django.contrib.auth import get_user_model

```

```

User = get_user_model()

```

```

# Or if you want to custom user model
from django.contrib.auth.models import AbstractUser

```

```

class User(AbstractUser):
    # add custom fields and methods

```

```

# To make Django use that model go to settings.py and add: AUTH_USER_MODEL = 'app_name.User'

```

Authentication: LoginView

LoginView is already pre-created by Django

```
from django.contrib.auth.views import LoginView
```

Add a url to reach that view

```
path('login/', LoginView.as_view(), name='login')
```

By default the LoginView will try to open a template name 'registration/login.html' and

Create a template under registration/login.html

```
{% extends "base.html" %}
```

```
{% block content %}
```

```
    <form method="post">
```

```
        {% csrf_token %}
```

```
        {{ form }}
```

```
        <button type="submit">Login</button>
```

```
    </form>
```

```
{% endblock content %}
```

When user click the Login button, the LoginView will try to authenticate the user with

If successful it will then login the user and redirect to LOGIN_REDIRECT_URL specified



Authentication: LogoutView

LogoutView is already pre-created by Django

```
from django.contrib.auth.views import LogoutView
```

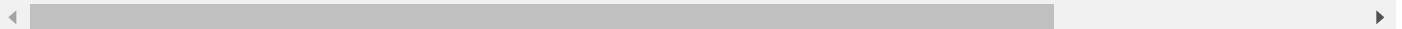
Add a url to reach that view

```
path('logout/', LogoutView.as_view(), name='logout')
```

Include a link in a template

```
<a href="{% url 'logout' %}">Logout</a>
```

After link is executed, the user will be logged out and redirect to LOGOUT_REDIRECT_URL specified



Authentication: SignupView

Create a SignupView (that view is not created by default)

import signupview form pre-created by Django

```
from django.contrib.auth.forms import UserCreationForm
```

```
from django.views.generic import CreateView
```

```
class SignupView(CreateView):
    template_name = 'registration/signup.html'
    form_class = UserCreationForm

    def get_success_url(self):
        return reverse("login")
```

```
# Create template: registration/signup.html
{% extends "base.html" %}
{% block content %}
    <form method="post">
        {% csrf_token %}
        {{ form }}
        <button type="submit">Signup</button>
    </form>
{% endblock content %}
```

Add a url to reach that view

```
from posts.views import SignupView
```

```
path('signup/', SignupView.as_view(), name='signup')
```

Optional: Customize the UserCreationForm

(forms.py)

```
from django.contrib.auth import get_user_model
from django.contrib.auth.forms import UserCreationForm
```

```
User = get_user_model()
```

```
class CustomUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username']
        fields_classes = {'username': UsernameField}
```

Optional pre-created authentication routes

urls.py

```
urlpatterns += path('', include('django.contrib.auth.urls'))
```

/login, /logout, /signup, etc.

Template Authentication helpers

Authentication links

```
<a href="{% url 'login' %}">Login</a>
<a href="{% url 'signup' %}">Signup</a>
<a href="{% url 'logout' %}">Logout</a>
```

Check if user login

```
{% if request.user.is_authenticated %}
    Logged in as: {{ request.user.username }}
{% endif %}
```

Authorization: LoginRequiredMixin and login_required

```
from django.contrib.auth.mixins import LoginRequiredMixin
```

Restrict views to auth user only (views.py)

```
class PostsCreateView(LoginRequiredMixin, generic.CreateView):
    ...
    ...
```

```
from django.contrib.auth.decorators import login_required
```

```
@login_required(login_url='/login')
def search_page(request):
    ...
    ...
```

Manual Authentication, Login and Logout

```
from django.contrib.auth import authenticate, login
```

```
def login_page(request):
    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect("index")

    return render(request, "registration/login.html", {})
```



```
def logout_page(request):  
    logout(request)  
    return redirect("index")
```

User Change password

```
# set_password will hash the password  
user.set_password('raw password')
```

Send Email

```
# settings.py  
EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"  
  
# Send email function  
from django.core.email import send_mail  
  
send_mail(  
    subject = "A new post has been created",  
    message = "Go to the web site to see the detail",  
    from_email = "test@test.com",  
    recipient_list = ["test2@text.com"]  
)
```

Signals

```
# models.py  
from django.db.models.signals import post_save, pre_save  
  
def post_user_created_signal(sender, instance, created, **kwargs):  
    if created:  
        UserProfile.objects.create(user=instance)  
  
# Launch the post_user_create_singal method if User model is save  
post_save.connect(post_user_created_signal, sender=User)
```

Seed

```
from .models import Product, Category  
from django.shortcuts import HttpResponseRedirect  
from faker import Faker
```

```
def seed(request):
    Product.objects.all().delete()
    Category.objects.all().delete()

    category = Category()
    category.name = "Sports"
    category.save()

    category = Category()
    category.name = "Home"
    category.save()

    fake = Faker()
    for _ in range(100):
        product = Product()
        product.name = fake.unique.word()
        product.short_description = fake.sentence()
        product.main_picture = fake.image_url()
        product.price = fake.random_digit() * 10
        product.category = Category.objects.order_by('?').first()
        product.save()

    return HttpResponse('Seeded')
```

.env key/value file

```
$ pip install django-dotenv
```

add code to manage.py file

```
import dotenv

def main():
    """Run administrative tasks."""
    dotenv.read_dotenv() #add this line
    ...
    ...
```

Create a file name '.env' in the root folder of your project

```
SECRET_KEY = 'your secret key'
ALLOWED_HOSTS=127.0.0.1
```

In settings.py change security related settings to point to the .env file

```
import os
SECRET_KEY = os.environ.get('SECRET_KEY')
ALLOWED_HOSTS = os.environ.get('ALLOWED_HOSTS')
```

Discussion (15)



Louis Madrigal • May 27



Nice article, Very helpful specially to those whos starting into python :)



Amir Hossein Roohi • Jul 25



oh damn, that faker just saved me a lot of time testing stuff, thanks



Dev Prakash • Apr 6



Brilliant work 🙌👍



Stephano Romain • Aug 6



Hi, nice article.
Just litle mess (dot repeated twice) on

Deleting

```
article = Article.objects.first()
article.delete()
```



Eric The Coder 🌟 • Aug 15



Thanks. Corrected