Django Tutorial for Beginners

Django is a high-level Python web framework that helps you build web applications quickly and securely.

1. Install Django

Make sure Python is installed:

```
python –version
```

Create a Virtual Environment

Windows

```
python -m venv venv
```

Activate the Virtual Environment

Windows (Command Prompt)

```
venv\Scripts\activate
```

Install Django:

```
pip install django
```

Verify installation:

```
django-admin --version
```

---

2. Create a Django Project

Create a new project:

```
django-admin startproject myproject
```

Move into the project directory:

```
cd myproject
```

Run the development server:

```
python manage.py runserver
```

Open your browser and visit:

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

You should see Django's welcome page.

---

3. Create an App

A Django project contains one or more apps.

Create an app:

```
python manage.py startapp blog
```

Project structure:

```
myproject/
│
├── manage.py
├── myproject/
│   ├── settings.py
│   ├── urls.py
│   └── ...
└── blog/
    ├── models.py
    ├── views.py
    └── ...
```

---

4. Register the App

Open `settings.py` and add the app:

```
INSTALLED_APPS = [
    ...
    'blog',
]
```

---

5. Create a View

In `blog/views.py`:

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, Django!")
```

---

6. Create URLs

Create `blog/urls.py`:

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

urlpatterns = [
    path('', views.home),
]
```

Connect it to the main project URL configuration.

In `myproject/urls.py`:

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

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

Run the server again:

```
python manage.py runserver
```

Visit:

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

You should see:

```
Hello, Django!
```

---

7. Create a Model

In `blog/models.py`:

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()

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

---

8. Create Database Tables

Generate migrations:

```
python manage.py makemigrations
```

Apply migrations:

```
python manage.py migrate
```

---

9. Use the Django Admin

Create an admin user:

```
python manage.py createsuperuser
```

Register the model in `blog/admin.py`:

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

admin.site.register(Post)
```

Run the server and visit:

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

Log in and manage posts through the admin interface.

---

10. Create a Template

Create:

`blog/templates/blog/home.html`

Add:

```
<!DOCTYPE html>
<html>
<head>
    <title>My Blog</title>
</head>
<body>
    <h1>Welcome to My Blog</h1>
</body>
</html>
```

Update `views.py`:

```
from django.shortcuts import render

def home(request):
    return render(request, 'blog/home.html')
```

---

11. Display Data from the Database

Update `views.py`:

```
from django.shortcuts import render
from .models import Post

def home(request):
    posts = Post.objects.all()
    return render(request, 'blog/home.html', {'posts': posts})
```

Update template:

```
<h1>Posts</h1>

{% for post in posts %}
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>
{% endfor %}
```

Django Models & ORM (Very Important)

Django uses ORM (Object Relational Mapping) to talk to databases without SQL.

Create a better model

```
# blog/models.py
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

Make migrations

```
python manage.py makemigrations
python manage.py migrate
```

Basic ORM queries

Open shell:

```
python manage.py shell
```

Try:

```
from blog.models import Post

Post.objects.all()
Post.objects.create(title="First", content="Hello")
Post.objects.filter(title="First")
Post.objects.get(id=1)
```

---

2. Django Admin (Make it Powerful)

Improve admin display

```
# blog/admin.py
from django.contrib import admin
from .models import Post

class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'created_at')
    search_fields = ('title',)

admin.site.register(Post, PostAdmin)
```

Now admin becomes searchable and cleaner.

---

3. Django Templates (Real UI)

Pass data to template

```
def home(request):
    posts = Post.objects.all()
    return render(request, "blog/home.html", {"posts": posts})
```

Template logic

```
<h1>My Blog</h1>

{% for post in posts %}
  <div>
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>
    <small>{{ post.created_at }}</small>
  </div>
{% endfor %}
```

---

4. Django Forms (User Input)

Create a form

```
# blog/forms.py
from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content']
```

Use form in view

```
from .forms import PostForm
from django.shortcuts import redirect

def create_post(request):
    form = PostForm(request.POST or None)

    if form.is_valid():
        form.save()
        return redirect('home')

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

Template(create.html)

```
<h1>Create Post</h1>

<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Save</button>
</form>
```

---

5. URLs with Names (Clean Routing)

```
# blog/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('create/', views.create_post, name='create'),
]
```

Use in template:

```
<a href="{% url 'create' %}">Create Post</a>
```

Static Files (CSS / JS)

settings.py

```
STATIC_URL = 'static/'
```

Folder structure

```
blog/
  static/
    blog/
      style.css
```

Use in template(html file code):-

```
{% load static %}
<link rel="stylesheet" href="{% static 'blog/style.css' %}">
```

full Django CRUD + static files tutorial. I'll give you a clean step-by-step project you can actually run.

We'll build a simple **Blog CRUD app** with:

- Create Post
- Read Posts
- Update Post
- Delete Post
- Static CSS styling

---

0. Create Project Setup

```
django-admin startproject mysite
cd mysite
python manage.py startapp blog
```

Add app in `settings.py`:

```
INSTALLED_APPS = [
    ...
    'blog',
]
```

---

1. Create Model (Database Table)

```
# blog/models.py
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

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

Run:

```
python manage.py makemigrations
python manage.py migrate
```

---

2. Create Forms (For Create & Update)

```
# blog/forms.py
from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content']
```

---

3. CRUD Views (Main Logic)

```
# blog/views.py
from django.shortcuts import render, redirect, get_object_or_404
from .models import Post
from .forms import PostForm
```

---

1. READ (List Posts)

```
def post_list(request):
    posts = Post.objects.all().order_by('-created_at')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

---

2. CREATE

```
def post_create(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('post_list')
    else:
        form = PostForm()

    return render(request, 'blog/post_form.html', {'form': form})
```

---

3. UPDATE

```
def post_update(request, id):

    post = get_object_or_404(Post, id=id)


    if request.method == "POST":

        form = PostForm(request.POST, instance=post)

        if form.is_valid():

            form.save()

            return redirect('post_list')

    else:

        form = PostForm(instance=post)


    return render(request, 'blog/post_form.html', {'form': form})
```

---

4. DELETE

```
def post_delete(request, id):
    post = get_object_or_404(Post, id=id)

    if request.method == "POST":
        post.delete()
        return redirect('post_list')

    return render(request, 'blog/post_delete.html', {'post': post})
```

---

4. URLs Setup

```
# blog/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('create/', views.post_create, name='post_create'),
    path('update/<int:id>/', views.post_update, name='post_update'),
    path('delete/<int:id>/', views.post_delete, name='post_delete'),
]
```

Main project:

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

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

---

5. Templates (UI)

Create folder:

```
blog/templates/blog/
```

---

1. Post List (READ)

```
<h1>All Posts</h1>

<a href="{% url 'post_create' %}">+ Create New</a>

{% for post in posts %}
  <div>
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>

    <a href="{% url 'post_update' post.id %}">Edit</a>
    <a href="{% url 'post_delete' post.id %}">Delete</a>
  </div>
{% endfor %}
```

---

2. Form Template (CREATE + UPDATE)

```
<h1>Post Form</h1>

<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Save</button>
</form>

<a href="{% url 'post_list' %}">Back</a>
```

---

3. DELETE Page

```
<h1>Are you sure?</h1>

<p>{{ post.title }}</p>

<form method="post">
  {% csrf_token %}
  <button type="submit">Yes Delete</button>
</form>

<a href="{% url 'post_list' %}">Cancel</a>
```

---

6. STATIC FILES (CSS Styling)

Step 1: settings.py

```
STATIC_URL = '/static/'
```

Step 2: Create folder

```
blog/static/blog/style.css
```

Step 3: Add CSS

```
body {
    font-family: Arial;
    background: #f4f4f4;
    margin: 20px;
}

div {
    background: white;
    padding: 10px;
    margin-bottom: 10px;
    border-radius: 8px;
}
```

Step 4: Use CSS in template

At top of HTML:

```
{% load static %}
<link rel="stylesheet" href="{% static 'blog/style.css' %}">
```

---

7. FINAL FLOW (How CRUD Works)

CREATE

Form → Save → DB

READ

DB → View → Template

UPDATE

Load object → edit → save

DELETE

Confirm → delete → redirect

---

8. RUN PROJECT

```
python manage.py runserver
```

Open:

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

Here's a **clean step-by-step Django project guide** to build your full system:

 Blog with Signup, Login, Logout, and User-owned CRUD

Follow exactly in order.

STEP 1: Create Project

```
django-admin startproject mysite
cd mysite
python manage.py startapp blog
```

STEP 2: Register App

Open `mysite/settings.py`

```
INSTALLED_APPS = [
    ...
    'blog',
]
```

STEP 3: Create Model (IMPORTANT)

Open `blog/models.py`

```
from django.db import models
from django.contrib.auth.models import User

class Post(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

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

□ STEP 4: Migrate Database

```
python manage.py makemigrations
python manage.py migrate
```

□ STEP 5: Create Forms

Create `blog/forms.py`

```
from django import forms
from .models import Post
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content']


class SignUpForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username', 'password1', 'password2']
```

□ STEP 6: Create Views (CORE LOGIC)

Open `blog/views.py`

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Post
from .forms import PostForm, SignUpForm
from django.contrib.auth.decorators import login_required
```

---

1. Signup

```
def signup(request):
    if request.method == "POST":
        form = SignUpForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = SignUpForm()

    return render(request, 'blog/signup.html', {'form': form})
```

---

2. LOGIN / LOGOUT handled by Django (no view needed)

---

3. READ (Home Page)

```
def post_list(request):
    posts = Post.objects.all().order_by('-created_at')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

---

4. CREATE (only logged-in user)

```
@login_required
def post_create(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.user = request.user
            post.save()
            return redirect('post_list')
    else:
        form = PostForm()

    return render(request, 'blog/post_form.html', {'form': form})
```

---

5. UPDATE (only owner)

```
from django.shortcuts import get_object_or_404, redirect, render
from django.contrib.auth.decorators import login_required

@login_required
def post_update(request, id):
post = get_object_or_404(Post, id=id, user=request.user)

form = PostForm(request.POST or None, instance=post)

if request.method == "POST" and form.is_valid():
form.save()
return redirect('post_list')

return render(request, 'blog/post_form.html', {'form': form})
```

---

6. DELETE (only owner)

```
@login_required
def post_delete(request, id):
    post = get_object_or_404(Post, id=id, user=request.user)

    if request.method == "POST":
        post.delete()
        return redirect('post_list')

    return render(request, 'blog/post_confirm_delete.html', {'post': post})
```

□ STEP 7: URLs Setup

Create `blog/urls.py`

```
from django.urls import path
from django.contrib.auth import views as auth_views
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),

    path('signup/', views.signup, name='signup'),

    path('login/',
auth_views.LoginView.as_view(template_name='blog/login.html'), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),

    path('create/', views.post_create, name='post_create'),
    path('update/<int:id>/', views.post_update, name='post_update'),
    path('delete/<int:id>/', views.post_delete, name='post_delete'),
]
```

Main `mysite/urls.py`

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

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

□ STEP 8: Settings (IMPORTANT)

Add this in `settings.py`

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/login/'
```

□ STEP 9: Create Templates

Create folder:

```
blog/templates/blog/
```

---

1. post_list.html

```
<h1>Posts</h1>

<a href="{% url 'post_create' %}">Create Post</a>
<a href="{% url 'logout' %}">Logout</a>

{% for post in posts %}
  <div>
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>

    {% if post.user == user %}
      <a href="{% url 'post_update' post.id %}">Edit</a>
      <a href="{% url 'post_delete' post.id %}">Delete</a>
    {% endif %}
  </div>
{% endfor %}
```

---

2. post_form.html

```
<h1>Post Form</h1>

<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Save</button>
</form>
```

---

3. post_delete.html

```
<h1>Delete Post</h1>

<p>{{ post.title }}</p>

<form method="post">
  {% csrf_token %}
  <button type="submit">Confirm Delete</button>
</form>
```

---

4. signup.html

```
<h1>Signup</h1>

<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Register</button>
</form>

<a href="{% url 'login' %}">Login</a>
```

---

5. login.html

```
<h1>Login</h1>

<form method="post">
  {% csrf_token %}
```

```
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>
```

□ STEP 10: Run Server

```
python manage.py runserver
```

Open:

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

□ FINAL RESULT

You now have:

Authentication

- Signup
- Login
- Logout

CRUD System

- Create post (user only)
- Read posts
- Update own post only
- Delete own post only

If you want **login/logout using Django AuthenticationForm (manual login view instead of LoginView)**. This is good for learning how auth really works.

I'll convert your project to **fully manual authentication system**.

□ 1. LOGIN using `AuthenticationForm`

views.py (custom login)

```
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, logout, authenticate
from django.shortcuts import render, redirect
```

□ LOGIN VIEW (manual)

```
def user_login(request):
    if request.method == "POST":
        form = AuthenticationForm(request, data=request.POST)

        if form.is_valid():
            user = form.get_user()    # authenticated user
            login(request, user)      # start session
            return redirect('post_list')
    else:
        form = AuthenticationForm()

    return render(request, 'blog/login.html', {'form': form})
```

□ 2. LOGOUT (manual view)

```
def user_logout(request):
    logout(request)
    return redirect('login')
```

□ 3. URLs update

Replace auth_views with your own views:

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

urlpatterns = [
    path('login/', views.user_login, name='login'),
    path('logout/', views.user_logout, name='logout'),
]
```

□ 4. LOGIN TEMPLATE (AuthenticationForm)

```
<h1>Login</h1>

<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>
```