

Let's create a **step-by-step beginner-friendly guide** to learning **Python Django** on a **Windows PC**.

1. What is Django?

- **Django** is a high-level **Python web framework** that encourages rapid development and clean, pragmatic design.
 - It helps you build **websites and web applications** quickly, without reinventing the wheel.
 - Key features:
 - Object-Relational Mapping (**ORM**) to interact with databases using Python code.
 - URL routing to map URLs to functions or views.
 - Templates for dynamically generating HTML.
 - Built-in admin interface.
 - Security features (against SQL injection, XSS, CSRF, etc.).
-

2. Setting Up Your Environment on Windows

Step 1: Install Python

1. Go to [Python official website](#).
2. Download the latest **Python 3.x** installer for Windows.
3. During installation:
 - Check **“Add Python to PATH”**
 - Choose **Install Now**.

☐ Tip: Open Command Prompt and type:

```
python --version
```

You should see something like `Python 3.12.x`.

Step 2: Install a Code Editor

- Recommended: **Visual Studio Code (VS Code)**
- Download from [VS Code website](#).
- Install Python extension in VS Code for code highlighting and debugging.

Step 3: Install Django

1. Open Command Prompt.
2. It's best to use a **virtual environment**:

```
python -m venv myenv  
myenv\Scripts\activate
```

You should see `(myenv)` before the prompt.

3. Install Django:

```
pip install django
```

4. Verify installation:

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

3. Creating Your First Django Project

1. Open Command Prompt in your desired folder.
2. Create a new project:

```
django-admin startproject mysite
```

3. Navigate to your project folder:

```
cd mysite
```

4. Run the development server:

```
python manage.py runserver
```

- Open your browser and go to `http://127.0.0.1:8000/`
 - You should see the **Django welcome page**.
-

4. Understanding Django Project Structure

```
mysite/
├── manage.py          # Command-line utility to interact with project
├── mysite/
│   ├── __init__.py
│   ├── settings.py   # Project configuration
│   ├── urls.py       # URL routing
│   ├── asgi.py       # ASGI entry-point
│   └── wsgi.py       # WSGI entry-point
```

- Tip: Treat `mysite/` as the **project container**, and you'll later create **apps** inside it.
-

5. Creating Your First App

1. Apps are like modules in Django.
2. Create an app:

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

- Folder structure:

```
blog/
├── admin.py
├── apps.py
├── models.py
├── tests.py
├── views.py
└── migrations/
```

3. Register the app in `mysite/settings.py` under `INSTALLED_APPS`:

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

6. Creating a Simple View and URL

1. In `blog/views.py`:

```
from django.http import HttpResponse

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

2. In `mysite/urls.py`:

```
from django.contrib import admin
from django.urls import path
from blog import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home), # Maps home page to our view
]
```

3. Run the server again:

```
python manage.py runserver
```

- Go to `http://127.0.0.1:8000/` → You should see **"Hello, Django Beginners!"**

7. Templates in Django

1. Create a folder `templates` inside the `blog` app:

```
blog/
├── templates/
│   └── home.html
```

2. In `home.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>My Django Site</title>
</head>
<body>
```

```
<h1>Welcome to Django!</h1>
</body>
</html>
```

3. Update `views.py`:

```
from django.shortcuts import render

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

- Refresh your browser to see the template rendered.
-

8. Using Models and Database

1. Open `blog/models.py`:

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

2. Apply migrations:

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

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

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

admin.site.register(Post)
```

4. Create a superuser to access admin:

```
python manage.py createsuperuser
```

- Login at <http://127.0.0.1:8000/admin/> to manage posts.

Let's dive into **Django Forms**. Forms are essential for accepting user input, like creating a blog post, registering a user, or submitting comments. I'll explain **step by step**, with examples that a beginner can follow on Windows.

1. What Are Django Forms?

Django forms are Python classes that let you:

- Render HTML form elements.
- Validate user input.
- Save data to the database (optional).

There are **two main types**:

1. **Form** – general-purpose forms not tied directly to a database.
 2. **ModelForm** – a form automatically linked to a Django model, so it can create or update records easily.
-

2. Creating a Simple Form

Step 1: Create a basic form

In `blog/forms.py`:

```
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(max_length=50, label="Your Name")
    email = forms.EmailField(label="Email Address")
    message = forms.CharField(widget=forms.Textarea, label="Message")
```

- `CharField` → text input
 - `EmailField` → validates email format
 - `TextArea` → multi-line input
-

Step 2: Create a view to handle the form

In `blog/views.py`:

```
from django.shortcuts import render
from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid(): # checks for correct input
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            message = form.cleaned_data['message']
            # Here you could save to database or send an email
            return render(request, 'contact_success.html', {'name': name})
    else:
        form = ContactForm()
    return render(request, 'contact.html', {'form': form})
```

Step 3: Create templates

`contact.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Contact Us</title>
</head>
<body>
    <h1>Contact Us</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Send</button>
    </form>
</body>
</html>
```

- `{{ form.as_p }}` renders all form fields wrapped in `<p>` tags.
- `{% csrf_token %}` protects against CSRF attacks (important!).

`contact_success.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Success</title>
</head>
<body>
    <h1>Thank you, {{ name }}! Your message has been sent.</h1>
```

```
</body>
</html>
```

Step 4: Add URL

In `mysite/urls.py`:

```
from blog import views

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

- Visit `http://127.0.0.1:8000/contact/` to see the form in action.
-

3. Using ModelForm to Submit Data to Database

If you want to save data directly to the database:

Step 1: Create a model

In `blog/models.py`:

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
```

Run migrations:

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

Step 2: Create a ModelForm

In `blog/forms.py`:

```
from django.forms import ModelForm
from .models import Post

class PostForm(ModelForm):
    class Meta:
```

```
model = Post
fields = ['title', 'content']
```

Step 3: Create a view to handle form submission

In `blog/views.py`:

```
def create_post(request):
    if request.method == 'POST':
        form = PostForm(request.POST)
        if form.is_valid():
            form.save() # saves to database
            return render(request, 'post_success.html', {'title':
form.cleaned_data['title']})
        else:
            form = PostForm()
            return render(request, 'create_post.html', {'form': form})
```

Step 4: Templates

create_post.html

```
<h1>Create a New Post</h1>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
</form>
```

post_success.html

```
<h1>Post "{{ title }}" created successfully!</h1>
```

Step 5: Add URL

```
path('create-post/', views.create_post),
```

Visit <http://127.0.0.1:8000/create-post/> and try submitting a post.

4. Key Tips for Beginners

1. Always use `{% csrf_token %}` in forms.
2. Use `is_valid()` to validate input before saving.
3. Use `cleaned_data` to safely access form data.
4. Start with simple `Form` before moving to `ModelForm`.
5. Test your forms by submitting different kinds of inputs to see validation in action.

Right now, your **contact view** collects the data but doesn't save it anywhere. To store the submitted data in a **database**, you need a **model** that matches the form fields, and then you save the form data into it.

Here's a **step-by-step guide** to modify your code so that it saves to the database.

1. Create a Model for Contact Messages

In `blog/models.py` (or whichever app you're using):

```
from django.db import models

class ContactMessage(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True) # automatically
    saves submission time

    def __str__(self):
        return f"{self.name} - {self.email}"
```

- `CharField` → stores text
 - `EmailField` → validates email
 - `TextField` → for longer text messages
 - `auto_now_add=True` → automatically sets the date/time when saved
-

2. Apply Migrations

After creating the model, run these commands in Command Prompt:

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

This will **create the database table** for `ContactMessage`.

3. Save Form Data to the Database

Update your `contact view` in `views.py`:

```

from django.shortcuts import render
from .forms import ContactForm
from .models import ContactMessage # import the model

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Get cleaned data from the form
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            message = form.cleaned_data['message']

            # Save data to database
            contact_message = ContactMessage(
                name=name,
                email=email,
                message=message
            )
            contact_message.save() # this writes it to the database

            # Pass name to success page
            return render(request, 'contact_success.html', {'name': name})
        else:
            form = ContactForm()
            return render(request, 'contact.html', {'form': form})

```

□ What changed:

- Imported `ContactMessage` model.
- Created an instance with `ContactMessage(name=name, email=email, message=message)`.
- Called `.save()` to store it in the database.

4. Optional: Using ModelForm (Simpler Way)

Instead of manually saving, you can use a **ModelForm** which automatically connects your form to the model.

```

# forms.py
from django.forms import ModelForm
from .models import ContactMessage

class ContactForm(ModelForm):
    class Meta:
        model = ContactMessage
        fields = ['name', 'email', 'message']

```

Then your `views.py` becomes simpler:

```
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            form.save() # automatically saves to database
            return render(request, 'contact_success.html', {'name':
form.cleaned_data['name']})
        else:
            form = ContactForm()
            return render(request, 'contact.html', {'form': form})
```

- No need to manually create the instance.
- Less code, easier to maintain.

Now, when someone submits the contact form, their **name, email, message, and timestamp** will be stored in the database!

Let's build a **complete beginner-friendly Django project** called “**Contact Manager**” where users can submit messages via a form, the messages are saved in the database, and you can view them in the Django admin. I'll give **step-by-step instructions** suitable for a Windows PC.

Django Contact Manager Project

1. Project Setup

1. Create a virtual environment and activate it:

```
python -m venv myenv
myenv\Scripts\activate
```

2. Install Django:

```
pip install django
```

3. Create a Django project:

```
django-admin startproject contact_manager
cd contact_manager
```

4. Create a new app called `contact`:

```
python manage.py startapp contact
```

5. Register the app in `contact_manager/settings.py`:

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

2. Create the Database Model

In `contact/models.py`:

```
from django.db import models

class ContactMessage(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.name} - {self.email}"
```

6. Apply migrations:

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

3. Enable Admin Access

1. In `contact/admin.py`:

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

admin.site.register(ContactMessage)
```

2. Create a superuser:

```
python manage.py createsuperuser
```

- Follow prompts to set username, email, and password.

3. Start the server:

```
python manage.py runserver
```

- Visit `http://127.0.0.1:8000/admin/`
 - Log in with your superuser account. You should see **Contact Messages** ready to view.
-

4. Create a ModelForm

In `contact/forms.py`:

```
from django.forms import ModelForm
from .models import ContactMessage

class ContactForm(ModelForm):
    class Meta:
        model = ContactMessage
        fields = ['name', 'email', 'message']
```

- This form is directly linked to your database model.
-

5. Create Views for the Form

In `contact/views.py`:

```
from django.shortcuts import render
from .forms import ContactForm

def contact_view(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            form.save() # saves to database automatically
            return render(request, 'contact_success.html', {'name':
form.cleaned_data['name']})
        else:
            form = ContactForm()
            return render(request, 'contact.html', {'form': form})
```

6. Create Templates

1. Inside `contact/` create a folder `templates/contact/`:

```
contact/
├── templates/
│   └── contact/
│       ├── contact.html
│       └── contact_success.html
```

contact.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Contact Us</title>
</head>
<body>
    <h1>Contact Us</h1>
    <form method="post">
```

```
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Send</button>
    </form>
</body>
</html>
```

contact_success.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Success</title>
</head>
<body>
    <h1>Thank you, {{ name }}! Your message has been sent.</h1>
    <a href="/contact/">Send another message</a>
</body>
</html>
```

7. Add URL Patterns

1. In contact/urls.py:

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

urlpatterns = [
    path('', views.contact_view, name='contact'),
]
```

2. In contact_manager/urls.py:

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

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

8. Run the Project

1. Start the server:

```
python manage.py runserver
```

2. Visit <http://127.0.0.1:8000/contact/>
 - o Submit a message via the form.
 - o You will see the **success page**.
 3. Check the **admin panel** at <http://127.0.0.1:8000/admin/>
 - o Your submitted messages are now in the database.
-

□ What You Learned in This Project

- Setting up a Django project and app on Windows.
 - Creating a **database model** (`ContactMessage`).
 - Creating a **ModelForm** linked to the model.
 - Submitting data via a **form** and saving it in the database.
 - Displaying a **success page**.
 - Viewing saved data in **Django admin**.
-

This is a **full beginner-friendly workflow** combining:

- URL routing
- Forms
- Models
- Templates
- Admin panel