```
/*****************************************************************
*  Django - COMPLETE MASTERY GUIDE (10-PAGE FORMAT)              *
*  GOAL: 100% knowledge, 10-page structure, readable, no waste.  *
*  ORDER: Most Used First -> Advanced Topics                     *
*****************************************************************/

/*
```

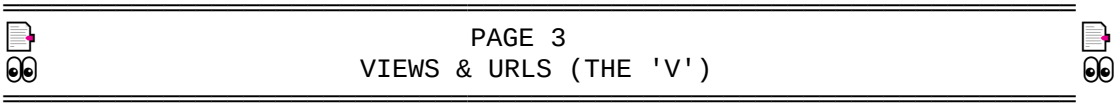
PAGE 1
INTRODUCTION & SETUP

```
*/
//--- WHAT IS DJANGO? ---
// Django ek high-level Python web framework hai jo rapid development aur clean
design encourage karta hai.
// Ye "batteries-included" hai, matlab isme authentication, admin panel jaisi
cheezein built-in aati hain.
// Architecture: MVT (Model-View-Template).

//--- INSTALLATION ---
// Terminal Command:
pip install Django

//--- CREATING A PROJECT ---
// Ek project poore web application ka container hota hai.
// Terminal Command:
django-admin startproject myproject .  // `.` current directory me project
banata hai.

//--- PROJECT STRUCTURE ---
// myproject/
//   - manage.py: Command-line utility.
//   - myproject/: Python package for your project.
//     - __init__.py
//     - settings.py: Project ki settings.
//     - urls.py: Project ke URL declarations.
//     - wsgi.py / asgi.py: Server entry-points.

//--- CREATING AN APP ---
// Ek app ek specific functionality handle karta hai (e.g., blog, polls).
// Terminal Command:
python manage.py startapp myapp

//--- REGISTERING THE APP ---
// `myproject/settings.py` me app ko `INSTALLED_APPS` me add karein.
INSTALLED_APPS = [
    # ...
    'myapp',
]

//--- RUNNING THE DEVELOPMENT SERVER ---
// Terminal Command:
python manage.py runserver

/*
```

PAGE 2
MODELS (THE 'M')

```
*/
// Model aapke data ka single, definitive source hai. Ye database table ko
represent karta hai.
// Models `myapp/models.py` me define hote hain.
```

```
//--- DEFINING A MODEL ---
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField('date published')

    def __str__(self):
        return self.title

//--- COMMON FIELD TYPES ---
// CharField: Text field (for small strings).
// TextField: Large text field.
// IntegerField, FloatField: Numbers.
// BooleanField: True/False.
// DateTimeField, DateField: Date and time.
// ForeignKey: Many-to-one relationship.
// ManyToManyField: Many-to-many relationship.
// OneToOneField: One-to-one relationship.

//--- MIGRATIONS ---
// Migrations aapke model changes ko database schema me apply karte hain.
// Step 1: Create migrations.
// Django aapke model changes ko detect karke migration file banata hai.
// Terminal Command:
python manage.py makemigrations myapp

// Step 2: Apply migrations.
// Ye migration file ko database par run karta hai.
// Terminal Command:
python manage.py migrate

/*
```




```
*/
// View ek function ya class hai jo web request leta hai aur web response return
karta hai.
// URLconfs URLs ko views se map karte hain.

//--- FUNCTION-BASED VIEWS (myapp/views.py) ---
from django.http import HttpResponse
from .models import Post

def index(request):
    latest_posts = Post.objects.order_by('-pub_date')[:5]
    output = ', '.join([p.title for p in latest_posts])
    return HttpResponse(output)

def detail(request, post_id):
    return HttpResponse(f"You're looking at post {post_id}.")

//--- URL CONFIGURATION (URLconf) ---
// Step 1: App URLconf (`myapp/urls.py` - create this file)
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'), // e.g., /myapp/
    path('<int:post_id>/', views.detail, name='detail'), // e.g., /myapp/5/
]
```

```
// Step 2: Project URLconf (`myproject/urls.py`)
// App ke URLconf ko project ke URLconf me include karein.
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')), // Include app's URLs.
]

/*
```



```
*/
// Template ek text file hai jo final output (HTML) ka structure define karta
hai.
// App-specific templates `myapp/templates/myapp/` me rakhein.

//--- USING TEMPLATES IN VIEWS (myapp/views.py) ---
from django.shortcuts import render

def index_template(request):
    latest_posts = Post.objects.order_by('-pub_date')[:5]
    context = {'latest_posts': latest_posts}
    return render(request, 'myapp/index.html', context)

//--- DJANGO TEMPLATE LANGUAGE (DTL) ---
// `myapp/templates/myapp/index.html`
/*
{% if latest_posts %}
    <ul>
    {% for post in latest_posts %}
        <li><a href="/myapp/{{ post.id }}/">{{ post.title }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No posts are available.</p>
{% endif %}
*/
// Variables: {{ post.title }}
// Tags: {% if ... %}, {% for ... %}
// Filters: {{ post.pub_date|date:"F j, Y" }}

//--- TEMPLATE INHERITANCE ---
// `base.html`
/*
<!DOCTYPE html>
<html>
<head><title>{% block title %}My Site{% endblock %}</title></head>
<body>
    {% block content %}{% endblock %}
</body>
</html>
*/
// `index.html`
/*
{% extends "myapp/base.html" %}
{% block title %}Home{% endblock %}
{% block content %}
    <h1>Latest Posts</h1>
    ...
```

```
{% endblock %}
*/

/*
```


PAGE 5
THE DJANGO ADMIN


```
*/
// Django ka admin interface ek powerful, ready-to-use feature hai data manage
karne ke liye.

//--- CREATING A SUPERUSER ---
// Admin me login karne ke liye ek superuser banayein.
// Terminal Command:
python manage.py createsuperuser

//--- REGISTERING MODELS ---
// Apne models ko admin me dikhane ke liye unhe register karein.
// `myapp/admin.py`
from django.contrib import admin
from .models import Post

admin.site.register(Post)

// Ab `http://127.0.0.1:8000/admin/` par jaakar aap Post objects
create/edit/delete kar sakte hain.

//--- CUSTOMIZING THE ADMIN ---
// Aap admin me models ka display customize kar sakte hain.
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'pub_date') // List view me fields dikhayein.
    list_filter = ['pub_date'] // Filter sidebar add karein.
    search_fields = ['title', 'content'] // Search box add karein.

admin.site.register(Post, PostAdmin) // Custom admin class ke saath register
karein.

/*
```


PAGE 6
FORMS


```
*/
// Django me forms handle karne ke liye powerful tools hain.

//--- CREATING A FORM CLASS (myapp/forms.py) ---
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(max_length=100)
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)

//--- HANDLING FORMS IN VIEWS (myapp/views.py) ---
from .forms import ContactForm
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Process the data in form.cleaned_data
            return HttpResponse('Thanks for contacting us!')
    else:
        form = ContactForm()
```

```
    return render(request, 'myapp/contact.html', {'form': form})

//--- RENDERING FORMS IN TEMPLATES (contact.html) ---
/*
<form action="/contact/" method="post">
    {% csrf_token %} <!-- CSRF Protection -->
    {{ form.as_p }} <!-- Renders form fields as <p> tags -->
    <input type="submit" value="Submit">
</form>
*/

//--- MODELFORMS ---
// ModelForms aapke Django model se automatically form generate karte hain.
from django.forms import ModelForm
from .models import Post

class PostForm(ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content'] // Model ke in fields se form banayein.

/*
```


PAGE 7
STATIC FILES & MEDIA


```
*/
//--- STATIC FILES (CSS, JS, Images) ---
// Static files wo hain jo change nahi hote (e.g., project's CSS).
// 1. `settings.py` me `STATIC_URL` set hota hai (`/static/`).
// 2. App-specific static files `myapp/static/myapp/` me rakhein.
// 3. `collectstatic` command saare static files ko `STATIC_ROOT` me collect
karta hai production ke liye.
//    `python manage.py collectstatic`

//--- USING STATIC FILES IN TEMPLATES ---
/*
{% load static %}
<link rel="stylesheet" href="{% static 'myapp/style.css' %}">
<img src="{% static 'myapp/my_image.png' %}" alt="My Image">
*/

//--- MEDIA FILES (User-uploaded files) ---
// Media files wo hain jo users upload karte hain.
// 1. `settings.py` me `MEDIA_URL` aur `MEDIA_ROOT` set karein.
//    MEDIA_URL = '/media/'
//    MEDIA_ROOT = BASE_DIR / 'media'
// 2. Project's `urls.py` me media URL pattern add karein (development ke liye).
/*
from django.conf import settings
from django.conf.urls.static import static
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
*/
// 3. Model me `FileField` ya `ImageField` use karein.
//    `avatar = models.ImageField(upload_to='avatars/')`

/*
```

PAGE 8
QUERYING THE DATABASE (ORM)


```
*/
// Django ka ORM (Object-Relational Mapper) se aap Python code se database query
```

kar sakte hain.

//--- RETRIEVING OBJECTS ---
// All objects: `Post.objects.all()`
// Get one object: `Post.objects.get(id=1)` (Raises DoesNotExist if not found)
// Filtering: `Post.objects.filter(pub_date__year=2023)`
// Excluding: `Post.objects.exclude(title__startswith='Old')`

//--- FIELD LOOKUPS ---
// `exact`: `Post.objects.get(id__exact=1)` (default)
// `iexact`: Case-insensitive match.
// `contains`: Case-sensitive containment.
// `icontains`: Case-insensitive containment.
// `in`: `Post.objects.filter(id__in=[1, 2, 3])`
// `gt`, `gte`, `lt`, `lte`: Greater than, less than, etc.
// `startswith`, `endswith`: String matching.

//--- CHAINING FILTERS ---
// `Post.objects.filter(pub_date__year=2023).exclude(title__startswith='Old')`

//--- COMPLEX LOOKUPS WITH Q OBJECTS ---
from django.db.models import Q
// `|` (OR), `&` (AND), `~` (NOT)
// `Post.objects.filter(Q(title__startswith='A') | Q(title__startswith='B'))`

//--- AGGREGATION ---
from django.db.models import Count, Avg
// `Post.objects.count()`
// `Post.objects.aggregate(Avg('likes'))`

/*


PAGE 9
USER AUTHENTICATION & PERMISSIONS

*/
// Django me ek complete user authentication system built-in hai.
// `django.contrib.auth` app `INSTALLED_APPS` me by default included hai.

//--- BUILT-IN VIEWS ---
// Django login, logout, password change ke liye views provide karta hai.
// Project's `urls.py` me inhe include karein:
// `path('accounts/', include('django.contrib.auth.urls')),`
// Ye `/accounts/login/`, `/accounts/logout/` jaise URLs add kar dega.

//--- LOGIN & LOGOUT ---
// `django.contrib.auth.views` se `LoginView` aur `LogoutView` use karein.
// Templates `registration/login.html` jaise paths par create karein.

//--- ACCESSING THE CURRENT USER ---
// View me: `request.user`
// Template me: `{{ user }}`
// `user.is_authenticated` check karta hai ki user logged in hai ya nahi.

//--- RESTRICTING ACCESS ---
// In views:
from django.contrib.auth.decorators import login_required
@login_required
def my_view(request):
    ...

// In Class-Based Views:
from django.contrib.auth.mixins import LoginRequiredMixin
class MyView(LoginRequiredMixin, View):

```
    ...

//--- PERMISSIONS ---
// Django me models par permissions (add, change, delete, view) automatically
create hote hain.
// `user.has_perm('myapp.add_post')`
// `@permission_required('myapp.add_post')`

/*
```



```
*/
//--- CLASS-BASED VIEWS (CBVs) ---
// Views ko classes ke roop me likhne ka ek powerful tarika.
from django.views.generic import ListView, DetailView
from .models import Post

class PostListView(ListView):
    model = Post
    template_name = 'myapp/post_list.html'

class PostDetailView(DetailView):
    model = Post
    template_name = 'myapp/post_detail.html'

//--- MIDDLEWARE ---
// Middleware ek framework hai jo Django ke request/response processing me hook
karta hai.
// Aap custom middleware `settings.py` ke `MIDDLEWARE` list me add kar sakte
hain.

//--- SIGNALS ---
// Signals allow certain senders to notify a set of receivers that some action
has taken place.
// Example: `post_save` signal model save hone ke baad fire hota hai.

//--- TESTING ---
// Django me testing ke liye built-in tools hain.
// `myapp/tests.py`
from django.test import TestCase
class MyModelTests(TestCase):
    def test_something(self):
        self.assertEqual(1 + 1, 2)

// Run tests: `python manage.py test`

//--- DEPLOYMENT ---
// 1. `settings.py` me `DEBUG = False` set karein.
// 2. `ALLOWED_HOSTS` set karein.
// 3. `python manage.py collectstatic` run karein.
// 4. Ek production-ready WSGI server jaise Gunicorn use karein.
//    `gunicorn myproject.wsgi`
// 5. Ek web server jaise Nginx ko reverse proxy ke roop me setup karein.

/************************************************************
*                     END OF MASTERY GUIDE                  *
************************************************************/
```