1. What is Django?

- High-level Python web framework.
- Batteries-included: ORM, authentication, admin panel, templates.
- Advantages: Rapid development, security, scalability, maintainable code.

2. Django Architecture

- Follows MTV pattern (Model-Template-View).
- Model: Database structure and logic.
- Template: Presentation layer (HTML).
- View: Business logic, processes requests, returns response.

3. Installing Django

```
pip install django
django-admin --version
```

Create project:

```
django-admin startproject myproject
cd myproject
python manage.py runserver
```

Create app:

```
python manage.py startapp myapp
```

4. Django Project Structure

```
myproject/
├── manage.py
├── myproject/
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── myapp/
    ├── models.py
    ├── views.py
    ├── urls.py
    ├── templates/
    ├── static/
    └── admin.py
```

5. Models (Database)

```
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    enrolled_on = models.DateField(auto_now_add=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Commands:

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

6. Admin Panel

```
from django.contrib import admin
from .models import Student
admin.site.register(Student)
```

Access: http://127.0.0.1:8000/admin

7. Views

Function-based view:

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Welcome to SmartSkills!")
```

Class-based view:

```
from django.views import View
from django.http import HttpResponse

class HomeView(View):
    def get(self, request):
        return HttpResponse("Welcome to SmartSkills!")
```

8. Templates

home.html

```
<h1>Welcome, {{ user.first_name }}!</h1>
<ul>
  {% for course in courses %}
    <li>{{ course.name }}</li>
  {% endfor %}
</ul>
```

Render in view:

```
from django.shortcuts import render

def home(request):
    courses = ["Python Basics", "Django", "AI for Beginners"]
    return render(request, "home.html", {"courses": courses})
```

9. URL Routing

Project-level urls.py:

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

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

App-level urls.py:

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

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

10. Forms and User Input

```
from django import forms

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

View:

```
def contact(request):
    if request.method == "POST":
        form = ContactForm(request.POST)
        if form.is_valid():
            return HttpResponse("Thank you!")
    else:
        form = ContactForm()
    return render(request, "contact.html", {"form": form})
```

11. Django REST Framework (APIs)

Install:

```
pip install djangorestframework
```

Serializer:

```
from rest_framework import serializers
from .models import Student

class StudentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Student
        fields = "__all__"
```

ViewSet:

```
from rest_framework import viewsets
from .models import Student
from .serializers import StudentSerializer
```

```python
class StudentViewSet(viewsets.ModelViewSet):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
```

URLs:

```python
from rest_framework import routers
from .views import StudentViewSet

router = routers.DefaultRouter()
router.register(r'students', StudentViewSet)
urlpatterns = router.urls
```

12. Authentication & Authorization

- Built-in `django.contrib.auth` for login, logout, registration.
- Example:

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

  @login_required
  def dashboard(request):
      return render(request, "dashboard.html")
  ```

13. Static & Media Files

Settings:

```python
STATIC_URL = '/static/'
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
```

14. Deployment Basics

- Gunicorn + Nginx + PostgreSQL
- Virtual environments: `venv` or `pipenv`
- Environment variables for secrets
- Optional: Docker for containerized deployment

15. Tips & Best Practices

- Use virtual environments per project.

- Follow DRY (Don't Repeat Yourself).
- Class-based views for cleaner code.
- Handle errors and log exceptions.
- Pagination for large datasets.
- Write unit tests.

Suggested Learning Path

1. Python fundamentals (OOP, lists, dicts, functions)
2. Small Django project (blog, todo, portfolio)
3. Database models & ORM
4. CRUD functionality
5. Authentication & permissions
6. REST APIs with Django REST Framework
7. Deployment on Heroku/AWS
8. Advanced topics: caching, async tasks, scaling

End of Django Notes