**1. Django Basics**

1. **What is Django, and why is it used?  
   Django is a Python framework used to build web applications quickly. It includes features like authentication, admin panel, and database management.**
2. **How does Django differ from Flask?**
   * **Django: Full-stack framework with built-in tools (admin panel, ORM).**
   * **Flask: Lightweight, minimal, and needs external libraries for extra features.**
3. **What is Django’s MVT architecture?**
   * **Model: Handles database operations.**
   * **View: Manages business logic and passes data to the template.**
   * **Template: Displays the data in HTML format.**
4. **Django’s main features:**
   * **ORM (Object-Relational Mapping)**
   * **Admin panel**
   * **Built-in authentication**
   * **URL routing**
   * **Middleware support**
   * **Security features**
5. **How does Django handle requests and responses?**
   * **The URL routes the request to the correct view.**
   * **The view processes the request and interacts with the model.**
   * **The response (HTML, JSON, etc.) is sent back to the user.**

**2. Models & Database**

1. **What is a Django Model?  
   A Django model is a Python class that defines database tables. The ORM converts it into SQL queries.**
2. **Advantages of Django ORM:**
   * **Simplifies database operations using Python code.**
   * **Automatically handles SQL queries.**
   * **Supports multiple databases.**
3. **Difference between ForeignKey, OneToOneField, and ManyToManyField:**
   * **ForeignKey: One-to-many relationship.**
   * **OneToOneField: One-to-one relationship.**
   * **ManyToManyField: Many-to-many relationship.**
4. **Difference between null=True and blank=True:**
   * **null=True: Allows NULL values in the database.**
   * **blank=True: Allows the field to be empty in forms.**
5. **How to make migrations:**
   * **python manage.py makemigrations: Creates migration files.**
   * **python manage.py migrate: Applies migrations to the database.**

**3. Views & Templates**

1. **Difference between FBV and CBV:**
   * **FBV (Function-Based View): Uses simple functions for logic.**
   * **CBV (Class-Based View): Uses classes with built-in methods.**
2. **How to pass data from views to templates:  
   Use the context dictionary to send data to templates.**
3. **Handling static and media files:**
   * **Static files: CSS, JS, images (via STATIC\_URL).**
   * **Media files: User uploads (via MEDIA\_URL).**
4. **Template tags and filters:**
   * **Tags: Control logic in templates ({% for %}, {% if %}).**
   * **Filters: Modify data display ({{ name|upper }}).**

**4. Forms & Validations**

1. **Difference between Django forms and model forms:**
   * **Forms: Manually create fields.**
   * **Model forms: Automatically generate fields from models.**
2. **How to validate form data:  
   Use is\_valid() method to check if the form data is valid.**
3. **How to handle file uploads:**
   * **Use enctype="multipart/form-data" in the form.**
   * **Use FileField or ImageField in models.**

**5. Authentication & Authorization**

1. **How does Django’s authentication system work?**
   * **Uses the built-in User model for login, logout, and password management.**
2. **How to create custom authentication:**
   * **Create a custom User model by extending AbstractUser or AbstractBaseUser.**
   * **Add it to settings.py.**
3. **Difference between permissions and groups:**
   * **Permissions: Allow specific actions (add, change, delete).**
   * **Groups: Collection of permissions for multiple users.**

**6. Middleware**

1. **What is middleware in Django?  
   Middleware is a layer between requests and responses. It processes data before sending it to the view.**
2. **How to create custom middleware:**
   * **Create a Python class with \_\_call\_\_() or process\_request() methods.**
   * **Add it to MIDDLEWARE in settings.py.**
3. **Built-in middleware examples:**
   * **SecurityMiddleware: Adds security features.**
   * **SessionMiddleware: Handles sessions.**
   * **AuthenticationMiddleware: Links users with requests.**

**7. Django REST Framework (DRF)**

1. **What is DRF, and why is it used?  
   DRF is a toolkit for building RESTful APIs in Django, making API development easier.**
2. **How to create an API:**
   * **Install DRF: pip install djangorestframework.**
   * **Create serializers and views.**
   * **Add DRF URLs in urls.py.**
3. **What are serializers?  
   Serializers convert Django models to JSON and vice versa.**
4. **Difference between ModelSerializer and Serializer:**
   * **ModelSerializer: Auto-generates fields from models.**
   * **Serializer: Manually defines fields.**
5. **What is token-based authentication?**
   * **Uses tokens instead of sessions.**
   * **Tokens are sent with requests for authentication.**

**8. Caching & Performance Optimization**

1. **How does caching work in Django?  
   Caching stores data temporarily to reduce database hits and improve speed.**
2. **Types of caching:**
   * **File-based caching**
   * **Database caching**
   * **Memcached**
   * **Local-memory caching**
3. **How to optimize database queries:**
   * **Use select\_related() and prefetch\_related() for better querying.**
   * **Apply caching.**
   * **Use database indexing.**

**9. Deployment & Security**

1. **How to deploy a Django project:**
   * **Use Gunicorn or uWSGI as the server.**
   * **Configure nginx or Apache as a reverse proxy.**
   * **Use Docker for containerization.**
2. **Django’s security features:**
   * **CSRF protection**
   * **XSS protection**
   * **SQL injection prevention**
   * **Clickjacking protection**
3. **How to prevent SQL Injection and XSS:**
   * **Use Django’s ORM to avoid SQL injection.**
   * **Use escape filters in templates to prevent XSS.**
4. **How to handle environment variables:**
   * **Use .env files with django-environ or os.environ in settings.py.**

**10. Miscellaneous**

1. **How does Django handle sessions?  
   Stores session data in the database or cache using SessionMiddleware.**
2. **Difference between get() and filter() in Django ORM:**
   * **get(): Returns a single object or raises an error if not found.**
   * **filter(): Returns a list of matching objects.**
3. **What are Django signals?  
   Signals allow different parts of the app to communicate. Examples: post\_save, pre\_save.**
4. **How to override Django admin functionalities:**
   * **Use admin.ModelAdmin.**
   * **Customize fields, filters, and templates.**
5. **What are pre\_save and post\_save signals?**
   * **pre\_save: Triggered before saving an object.**
   * **post\_save: Triggered after saving an object.**