Basic Level Questions:

1. **What is Django?**

   Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is open-source and "batteries-included," providing built-in tools for handling databases, forms, authentication, and other web application needs.
2. **What is the Django MVT architecture?**

   Django follows the Model-View-Template (MVT) architecture:
   - **Model: Defines the structure of the database.**
   - **View: Handles the logic and processes HTTP requests.**
   - **Template: Represents the presentation layer, where HTML files display the data from views.**
3. **How do you create a Django project and app?**
   - **To create a Django project: django-admin startproject projectname**
   - **To create a Django app: python manage.py startapp appname**
4. **What is settings.py used for?**

   settings.py is the configuration file for the Django project. It includes database settings, installed apps, static files, middleware, templates, and other project-wide settings.
5. **How does Django handle requests?**

   Django processes requests through URL routing to views. When a URL is requested, Django matches it against URL patterns defined in urls.py, routes it to the corresponding view, and returns the HTTP response generated by the view.

6. **What is a QuerySet?**

   A QuerySet is a collection of database queries to retrieve objects from your database. It allows filtering, ordering, and chaining to modify the data retrieved.

7. **What are Django models?**

   Django models are Python classes that map to database tables. They define the structure of the data, fields, and behavior of data stored in the database, with ORM managing interactions.

8. **How does Django handle forms?**

   Django simplifies form handling with forms.py. Forms handle validation, input processing, and error reporting, and can be used to handle data from both GET and POST requests.

9. **What is a virtual environment, and why do we use it in Django?**

   A virtual environment isolates dependencies for each project, preventing conflicts between different packages and versions.

10. **What is an HttpResponse?**

    HttpResponse is a Django class for returning data to the client as an HTTP response. Other response classes include JsonResponse (for JSON data) and HttpResponseRedirect (for redirects).

Intermediate Level Questions:

11. **Explain Django ORM (Object-Relational Mapping).**

    Django ORM lets developers interact with databases using Python code instead of raw SQL. With ORM, you create models representing database tables, allowing database interactions through Python objects.

12. **What is ForeignKey in Django models?**

    ForeignKey is a type of field that creates a one-to-many relationship between models, often used to link related data across tables.

13. **How do you handle file uploads in Django?**

    Use FileField or ImageField in models to handle file uploads, specifying where files should be stored in settings.py using MEDIA_ROOT and MEDIA_URL.

14. **Explain the concept of middleware in Django.**

    Middleware is a stack of components that process requests and responses. Common middleware includes security (e.g., CSRF protection), session management, and user authentication.

15. **What are Django signals?**

    Signals allow decoupled applications to get notified when certain actions occur. Common signals include post_save (triggered after saving an instance) and post_delete (triggered after deleting an instance).

16. **How does Django handle static and media files?**

    Static files (CSS, JavaScript) are served via STATIC_URL and configured in settings.py with STATICFILES_DIRS. Media files (user-uploaded content) are served via MEDIA_URL and saved in MEDIA_ROOT.

17. **What are migrations in Django?**

    Migrations are changes applied to the database schema based on model changes. Django manages migrations using python manage.py makemigrations and python manage.py migrate.

18. **How do you implement authentication in Django?**

    Django provides a built-in authentication system with login, logout, and registration, using the User model and views for managing user access.

19. **What is Django's request object?**

    request is an HTTPRequest object that contains metadata about the HTTP request, such as request.GET, request.POST, and request.user.

20. **How can you secure a Django application?**

    Security features include CSRF protection, XSS prevention, session management, and using HTTPS. SECURE_SSL_REDIRECT, X_FRAME_OPTIONS, and SECURE_HSTS_SECONDS in settings.py improve security.

Advanced Level Questions:

21. **What is Django's caching framework, and how does it improve performance?**

    Django caching stores data temporarily to reduce database hits and load times. Cache backends like Memcached and Redis are supported, and caching can be applied to views, database queries, and templates.

22. **How does Django handle asynchronous tasks?**

    Django can handle asynchronous tasks using tools like Celery, which manages background tasks, while recent versions support async views for non-blocking operations.

23. **What are class-based views (CBVs), and how are they different from function-based views (FBVs)?**

    CBVs use classes to encapsulate logic, while FBVs are simple functions. CBVs are reusable and modular, making them ideal for complex views that benefit from inheritance.

24. **How does Django's signals system work internally?**

    Django's signal-dispatching mechanism lets you connect custom functions to signal events like post_save to trigger actions after saving a model.

25. **What is Django REST Framework (DRF), and why use it?**

    DRF is a toolkit for building APIs in Django, providing serializers, viewsets, and authentication features, making it easier to develop RESTful web services.

26. **How do you optimize database queries in Django?**

    Optimizations include using select_related (for foreign keys), prefetch_related (for many-to-many), and only or defer (to limit fields fetched).

27. **What is the purpose of the __str__ method in Django models?**

    __str__ provides a human-readable name for each model instance, making it easier to read in the Django admin or debugging output.

28. **How would you handle multi-tenancy in Django?**

    Multi-tenancy can be handled by separating data by schema or database for each tenant, or by using shared tables with tenant-specific filtering.

29. **Explain the difference between sessions and cookies.**

    Sessions store user data on the server, while cookies store data on the client's browser. Django handles sessions securely with session keys stored in cookies.

30. **How do you manage API rate limiting in Django?**

    Rate limiting can be implemented using Django middleware or with Django REST Framework's throttle_classes for more fine-grained control over API usage.