Top 75 Django Interview Questions for 2025

July 2025

Introduction

This document provides 75 Django interview questions with answers and code examples, organized into Basics, Intermediate, and Advanced sections. Designed for 2025 technical interviews, it helps developers prepare for roles requiring Django expertise. Each question includes a practical code snippet to illustrate the concept.

1 Django Basics

1. **What is Django, and what are its core components?**
   Answer: Django is a Python web framework for rapid development, emphasizing security and scalability. Core components include ORM, URL dispatcher, views, and templates.

```
# settings.py
INSTALLED_APPS = ['django.contrib.admin',
    'django.contrib.auth']
```

2. **How do you start a new Django project?**
   Answer: Use `django-admin startproject <project_name>` to create a project directory.

```
# Terminal
django-admin startproject myproject
```

3. **What is the purpose of `manage.py`?**
   Answer: `manage.py` is a command-line utility for tasks like running the server or migrations.

```
# Terminal
python manage.py runserver
```

4. **How do you create a Django app?**
   Answer: Run `python manage.py startapp <app_name>` and add it to `INSTALLED_APPS`.

```
# Terminal
python manage.py startapp blog

# settings.py
INSTALLED_APPS = ['blog']
```

5. **What is a Django model?**
   Answer: A model is a Python class defining database structure and behavior.

```
# blog/models.py
from django.db import models
class Post(models.Model):
    title = models.CharField(max_length=200)
```

6. **How do you apply migrations in Django?**
   Answer: Run `makemigrations` to create migration files and `migrate` to apply them.

```
# Terminal
python manage.py makemigrations
python manage.py migrate
```

7. **What is the Django Admin Interface?**
   Answer: A web-based tool for managing model data, configured in `admin.py`.

```
# blog/admin.py
from django.contrib import admin
from .models import Post
admin.site.register(Post)
```

8. **How do you define URL patterns in Django?**
   Answer: Use `path()` in `urls.py` to map URLs to views.

```
# blog/urls.py
from django.urls import path
from . import views
urlpatterns = [path('posts/', views.post_list,
   name='post_list')]
```

9. **What is a Django view?**
   Answer: A view processes HTTP requests and returns responses.

```
# blog/views.py
from django.http import HttpResponse
def post_list(request):
    return HttpResponse("List of posts")
```

10. **How do you render templates in Django?**
Answer: Use `render()` to combine templates with context data.

```
# blog/views.py
from django.shortcuts import render
def home(request):
    return render(request, 'blog/home.html', {'title':
        'Home'})
```

11. **How do you configure templates in Django?**
Answer: Set `TEMPLATES['DIRS']` in `settings.py` for template directories.

```
# settings.py
TEMPLATES = [{'BACKEND':
   'django.template.backends.django.DjangoTemplates', 'DIRS':
   ['templates']}]
```

12. **How do you handle static files in Django?**
Answer: Configure `STATIC_URL` and `STATICFILES_DIRS`, use `% static %` in templates, and run `collectstatic` for production.

```
# settings.py
STATIC_URL = '/static/'
STATICFILES_DIRS = ['static']
STATIC_ROOT = 'staticfiles'

# Terminal
python manage.py collectstatic
```

13. **What is Django's CSRF protection?**
Answer: CSRF tokens prevent cross-site request forgery in forms.

```
<!-- blog/templates/form.html -->
<form method="post">{% csrf_token %}
    <input type="submit">
</form>
```

14. **How do you create a superuser in Django?**
Answer: Run `createsuperuser` to create an admin user.

```
# Terminal
python manage.py createsuperuser
```

15. **What is a Django QuerySet?**
Answer: A QuerySet is a lazily evaluated collection of database queries.

```python
# blog/views.py
from .models import Post
posts = Post.objects.all()
```

16. **How do you filter QuerySets in Django?**
Answer: Use `filter()` to query objects based on conditions.

```python
# blog/views.py
posts = Post.objects.filter(published=True)
```

17. **What is the `exclude()` method?**
Answer: `exclude()` filters out objects matching criteria.

```python
# blog/views.py
posts = Post.objects.exclude(published=False)
```

18. **How do you order QuerySets?**
Answer: Use `order_by()` to sort QuerySets, with `-` for descending order.

```python
# blog/views.py
from .models import Post
posts = Post.objects.order_by('-created_at')
```

19. **What is the `get()` method?**
Answer: `get()` retrieves a single object or raises an exception.

```python
# blog/views.py
post = Post.objects.get(id=1)
```

20. **How do you create a Django form?**
Answer: Define a form class and render it in a template.

```python
# blog/forms.py
from django import forms
class PostForm(forms.Form):
    title = forms.CharField(max_length=200)
```

21. **What are Django template tags?**
Answer: Tags provide logic in templates, like loops or conditionals.

```
<!-- blog/templates/post_list.html -->
{% for post in posts %}
    <p>{{ post.title }}</p>
{% endfor %}
```

22. **How do you handle POST requests in Django?**
Answer: Check `request.method` and process `request.POST`.

```python
# blog/views.py
from .forms import PostForm
def create_post(request):
    if request.method == 'POST':
        form = PostForm(request.POST)
        if form.is_valid():
            form.save()
```

23. **What is Django's authentication system?**
Answer: Manages users and permissions via `django.contrib.auth`.

```python
# blog/views.py
from django.contrib.auth import login
def login_view(request):
    login(request, user)
```

24. **How do you redirect in Django?**
Answer: Use `redirect()` to navigate to another URL.

```python
# blog/views.py
from django.shortcuts import redirect
def my_view(request):
    return redirect('post_list')
```

25. **What is the `DEBUG` setting?**
Answer: `DEBUG=True` shows detailed errors; set to `False` in production.

```python
# settings.py
DEBUG = False
```

2 Django Intermediate

1. **What is a Django model manager?**
Answer: A manager provides QuerySet methods for a model, customizable for queries.

```python
# blog/models.py
class PostManager(models.Manager):
    def published(self):
        return self.filter(published=True)
class Post(models.Model):
    objects = PostManager()
```

2. **How do you create a custom model manager?**
   Answer: Subclass `models.Manager` and assign to `objects`.

```python
# blog/models.py
from django.db import models
class CustomManager(models.Manager):
    def active(self):
        return self.filter(is_active=True)
class Post(models.Model):
    objects = CustomManager()
```

3. **What is a `ForeignKey`?**
   Answer: Defines a one-to-many relationship linking to another model's primary key.

```python
# blog/models.py
class Comment(models.Model):
    post = models.ForeignKey('Post', on_delete=models.CASCADE)
```

4. **What is a `ManyToManyField`?**
   Answer: Creates a many-to-many relationship using a junction table.

```python
# blog/models.py
class Post(models.Model):
    tags = models.ManyToManyField('Tag')
```

5. **What is `select_related()`?**
   Answer: Performs a SQL JOIN to fetch ForeignKey objects in one query.

```python
# blog/views.py
from .models import Comment
comments = Comment.objects.select_related('post').all()
```

6. **What is `prefetch_related()`?**
   Answer: Fetches ManyToMany or reverse ForeignKey objects in separate queries.

```python
# blog/views.py
from .models import Post
posts = Post.objects.prefetch_related('tags').all()
```

7. **How do you create a class-based view?**
   Answer: Subclass `View` or a generic view like `ListView`.

```python
# blog/views.py
from django.views.generic import ListView
class PostListView(ListView):
    model = Post
    template_name = 'post_list.html'
```

8. **What are Django signals?**
   Answer: Trigger functions on events like model saves.

```python
# blog/signals.py
from django.db.models.signals import post_save
from django.dispatch import receiver
@receiver(post_save, sender='blog.Post')
def post_saved(sender, instance, **kwargs):
    print("Post saved")
```

9. **How do you create a custom signal?**
   Answer: Define a `Signal` and connect to a receiver.

```python
# blog/signals.py
from django.dispatch import Signal
my_signal = Signal()
@receiver(my_signal)
def my_handler(sender, **kwargs):
    print("Custom signal")
```

10. **What is Django middleware?**
    Answer: Processes requests and responses globally.

```python
# blog/middleware.py
def log_middleware(get_response):
    def middleware(request):
        print("Request received")
        return get_response(request)
    return middleware
```

11. **How do you create custom middleware?**
    Answer: Define a class and add to `MIDDLEWARE`.

```python
# settings.py
MIDDLEWARE = ['blog.middleware.LogMiddleware']
```

12. **What is Django's caching framework?**
    Answer: Stores data to reduce database load.

```python
# blog/views.py
from django.views.decorators.cache import cache_page
@cache_page(60 * 10)
def post_list(request):
    return render(request, 'post_list.html')
```

13. **How do you handle file uploads in Django?**
    Answer: Use `FileField` and process `request.FILES`.

```python
# blog/models.py
class Document(models.Model):
    file = models.FileField(upload_to='uploads/')
# blog/views.py
def upload(request):
    if request.method == 'POST':
        form = DocumentForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
```

14. **What is the Django Debug Toolbar?**
Answer: A tool for profiling queries and performance.

```python
# settings.py
INSTALLED_APPS = ['debug_toolbar']
MIDDLEWARE =
   ['debug_toolbar.middleware.DebugToolbarMiddleware']
```

15. **How do you implement pagination in Django?**
Answer: Use `Paginator` to split QuerySets into pages.

```python
# blog/views.py
from django.core.paginator import Paginator
def post_list(request):
    posts = Post.objects.all()
    paginator = Paginator(posts, 10)
    page = paginator.get_page(request.GET.get('page'))
    return render(request, 'post_list.html', {'page': page})
```

16. **What are template filters?**
Answer: Transform template variables, like `upper`.

```
<!-- blog/templates/post.html -->
{{ title|upper }}
```

17. **How do you create a custom template filter?**
Answer: Define a function and register with `@register.filter`.

```python
# blog/templatetags/filters.py
from django import template
register = template.Library()
@register.filter
def add_prefix(value, arg):
    return f"{arg}{value}"
```

18. **How do you extend the Django `User` model?**
Answer: Subclass `AbstractUser` or use a one-to-one field.

```python
# blog/models.py
from django.contrib.auth.models import AbstractUser
class CustomUser(AbstractUser):
    bio = models.TextField()
```

19. **What is `LoginRequiredMixin`?**
Answer: Restricts views to authenticated users.

```python
# blog/views.py
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views import View
class SecureView(LoginRequiredMixin, View):
    def get(self, request):
        return HttpResponse("Secure")
```

20. **How do you handle database transactions?**
Answer: Use `atomic()` for atomic operations.

```python
# blog/views.py
from django.db import transaction
@transaction.atomic
def update_post(request, pk):
    post = Post.objects.get(pk=pk)
    post.title = 'Updated'
    post.save()
```

21. **What is the contenttypes framework?**
Answer: Enables generic relationships using `ContentType`.

```python
# blog/models.py
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import
   GenericForeignKey
class Comment(models.Model):
    content_type = models.ForeignKey(ContentType,
       on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey()
```

22. **How do you implement internationalization?**
Answer: Use `gettext` and `makemessages` for translations.

```python
# blog/views.py
from django.utils.translation import gettext as _
def greet(request):
    return HttpResponse(_("Hello"))
```

23. **What are generic views?**
Answer: Pre-built views for common tasks, like `ListView`.

```
# blog/views.py
from django.views.generic import ListView
class PostListView(ListView):
    model = Post
    template_name = 'post_list.html'
```

24. **How do you secure a Django application?**
Answer: Use HTTPS, CSRF tokens, and set `DEBUG=False`.

```
# settings.py
SECURE_SSL_REDIRECT = True
DEBUG = False
```

25. **How do you create a custom admin interface?**
Answer: Subclass `ModelAdmin` to add custom fields and actions.

```
# blog/admin.py
from django.contrib import admin
from .models import Post
class PostAdmin(admin.ModelAdmin):
    list_display = ['title', 'created_at']
admin.site.register(Post, PostAdmin)
```

3 Django Advanced

1. **How do you optimize Django querysets?**
Answer: Use `select_related()`, `prefetch_related()`, and `only()` to reduce database queries.

```
# blog/views.py
posts =
   Post.objects.select_related('author').prefetch_related('tags').only('t
   'author__name')
```

2. **What are Django's `Q` objects?**
Answer: `Q` objects enable complex queries with logical operators.

```
# blog/views.py
from django.db.models import Q
posts = Post.objects.filter(Q(title__contains='django') |
   Q(published=True))
```

3. **How do you use Django's `F` expressions?**
   Answer: `F` expressions reference field values in queries.

```python
# blog/views.py
from django.db.models import F
Post.objects.update(views=F('views') + 1)
```

4. **What is Django's database connection pooling?**
   Answer: Manages reusable database connections for efficiency.

```python
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'my_db',
        'CONN_MAX_AGE': 600,
    }
}
```

5. **How do you implement custom model fields?**
   Answer: Subclass `Field` to define custom data types.

```python
# blog/models.py
from django.db import models
class JSONField(models.Field):
    def db_type(self, connection):
        return 'jsonb'
class Post(models.Model):
    data = JSONField()
```

6. **What is Django's `Aggregate` function?**
   Answer: Computes summary values like `Count` or `Sum`.

```python
# blog/views.py
from django.db.models import Count
post_count = Post.objects.aggregate(total=Count('id'))
```

7. **How do you use `annotate` in Django?**
   Answer: Adds calculated fields to QuerySet objects.

```python
# blog/views.py
from django.db.models import Count
posts = Post.objects.annotate(comment_count=Count('comments'))
```

8. **How do you handle database migrations with zero downtime?**
   Answer: Apply migrations in phases, ensuring backward compatibility.

```python
# blog/migrations/0002_add_field.py
from django.db import migrations, models
class Migration(migrations.Migration):
    dependencies = [('blog', '0001_initial')]
    operations = [migrations.AddField(model_name='Post',
       name='new_field',
       field=models.CharField(max_length=100, null=True))]
```

9. **What is Django's `TransactionTestCase`?**
   Answer: Tests database operations with transaction rollback.

```python
# blog/tests.py
from django.test import TransactionTestCase
class PostTests(TransactionTestCase):
    def test_create_post(self):
        Post.objects.create(title='Test')
        self.assertEqual(Post.objects.count(), 1)
```

10. **How do you implement custom template tags?**
    Answer: Define a function and register with `simple_tag`.

```python
# blog/templatetags/tags.py
from django import template
register = template.Library()
@register.simple_tag
def current_year():
    from datetime import date
    return date.today().year
```

11. **What is Django's `check` framework?**
    Answer: Validates project configuration during deployment.

```python
# blog/checks.py
from django.core.checks import register, Warning
@register()
def example_check(app_configs, **kwargs):
    errors = []
    if settings.DEBUG:
        errors.append(Warning('DEBUG should be False in
           production'))
    return errors
```

12. **How do you use Django with PostgreSQL full-text search?**
    Answer: Use `SearchVector` and `SearchQuery` for text search.

```python
# blog/views.py
from django.contrib.postgres.search import SearchVector
```

```
posts =
    Post.objects.annotate(search=SearchVector('title')).filter(search='dja
```

13. **How do you implement async views in Django?**
Answer: Use `async def` with async-compatible middleware.

```
# blog/views.py
from django.http import HttpResponse
async def async_view(request):
    return HttpResponse("Async response")
```

14. **What is Django's `defer` method?**
Answer: Defers loading of specified fields to reduce query overhead.

```
# blog/views.py
posts = Post.objects.defer('content')
```

15. **How do you handle concurrent updates in Django?**
Answer: Use `select_for_update()` for atomic updates.

```
# blog/views.py
from django.db.models import F
Post.objects.select_for_update().filter(id=1).update(views=F('views')
    + 1)
```

16. **What is Django's `Manager.raw()` query?**
Answer: Executes raw SQL queries via `raw()`.

```
# blog/views.py
posts = Post.objects.raw('SELECT * FROM blog_post WHERE
    published = %s', [True])
```

17. **How do you profile Django applications?**
Answer: Use `django-silk` or `Debug Toolbar` to analyze queries.

```
# settings.py
INSTALLED_APPS = ['silk']
MIDDLEWARE = ['silk.middleware.SilkyMiddleware']
```

18. **How do you implement custom validators in Django?**
Answer: Define a function and apply to model fields.

```
# blog/models.py
from django.core.exceptions import ValidationError
def validate_title(value):
    if len(value) < 5:
```

```
        raise ValidationError('Title too short')
class Post(models.Model):
    title = models.CharField(max_length=200,
       validators=[validate_title])
```

19. **What is Django's `Subquery` expression?**
Answer: Embeds queries within queries for advanced filtering.

```
# blog/views.py
from django.db.models import Subquery
comments =
   Comment.objects.filter(post_id__in=Subquery(Post.objects.filter(publis
```

20. **How do you use Django with Redis for sessions?**
Answer: Configure `session backend` to use Redis.

```
# settings.py
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
    }
}
```

21. **What is Django's event loop integration?**
Answer: Supports async tasks using Python's event loop.

```
# blog/views.py
import asyncio
from django.http import HttpResponse
async def async_task(request):
    await asyncio.sleep(1)
    return HttpResponse("Async task completed")
```

22. **How do you implement custom authentication backends?**
Answer: Subclass `ModelBackend` to define custom authentication.

```
# blog/auth.py
from django.contrib.auth.backends import ModelBackend
from django.contrib.auth.models import User
class CustomBackend(ModelBackend):
    def authenticate(self, request, email=None,
       password=None, **kwargs):
        try:
            user = User.objects.get(email=email)
            if user.check_password(password):
                return user
```

```
        except User.DoesNotExist:
            return None
```

23. **What is Django's `Window` class?**
Answer: Performs window functions for ranking or aggregation.

```
# blog/views.py
from django.db.models.expressions import Window
from django.db.models.functions import Rank
posts = Post.objects.annotate(rank=Window(expression=Rank(),
    order_by='-rating'))
```

24. **How do you implement database sharding in Django?**
Answer: Route queries to different databases using a custom router.

```
# settings.py
DATABASES = {
    'default': {'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'default.db'},
    'shard1': {'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'shard1.db'}
}
# blog/db_router.py
class ShardRouter:
    def db_for_read(self, model, **hints):
        return 'shard1' if model._meta.app_label == 'blog'
            else 'default'
    def db_for_write(self, model, **hints):
        return self.db_for_read(model, **hints)
```

25. **How do you deploy Django with Docker?**
Answer: Use a Dockerfile to containerize Django and its dependencies.

```
# Dockerfile
FROM python:3.11
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "myproject.wsgi:application", "--bind", "0.0.0.0:8000"]
```

Conclusion

These 75 Django questions with code examples prepare you for 2025 technical interviews. Practice these snippets and consult Django's documentation for deeper insights.