Python & Django Interview Questions with Answers

■ Python - Beginner Level

What are Python's key features?

Easy to learn, interpreted, object-oriented, portable, huge library.

What is PEP 8?

Python Enhancement Proposal defining style guidelines for consistent, readable code.

Mutable vs Immutable

Mutable: list, dict, set. Immutable: tuple, str, int.

Lists, Tuples, Sets, Dicts

List=ordered mutable, Tuple=ordered immutable, Set=unique unordered, Dict=key-value.

Memory Management

Managed by private heap and garbage collector.

is vs ==

`is` checks identity, `==` checks equality.

Decorators

Functions that modify other functions using @ syntax.

List Comprehensions

Compact way to create lists, e.g. [x*x for x in range(5)].

with Statement

Manages resources automatically, e.g. with open(file) as f.

Modules & Packages

Modules=single files, Packages=directories with __init__.py.

■ Python - Intermediate Level

Shallow vs Deep Copy

Shallow copies references, deep copies recursively.

Garbage Collection

Uses reference counting and cyclic GC.

Generators & Iterators

Iterators use __next__, generators use yield.

Decorators @staticmethod/@classmethod/@property

Control class behavior and access.

GIL

Only one thread executes bytecode at a time.

__init__ vs __new__

__new__ creates object, __init__ initializes it.

Exception Handling

try/except/finally blocks.

Context Managers

Manage setup/teardown with __enter__ and __exit__.

Dependency Management

Use requirements.txt, Pipenv, Poetry.

Duck Typing

Behavior defines type, not inheritance.

■ Python - Advanced Level

Multithreading

threading module; limited by GIL.

Async & Await

For async non-blocking operations using asyncio.

Optimizing Slow Script

Profile, optimize algorithms, use caching, NumPy, Cython.

Metaclasses

Classes controlling class creation.

Memory Model

Stack for calls, heap for objects.

@dataclass

Simplifies class creation with auto methods.

Type Hints

Provide static type info for clarity.

__slots__

Restricts attributes to save memory.

Serialization

pickle/json for data persistence.

Monkey Patching

Modify classes/modules at runtime.

■ Django - Beginner Level

What is Django?

A high-level Python web framework using MVT.

Django Features

ORM, Admin, Templates, Middleware, Security.

MVT Architecture

Model=DB, View=Logic, Template=UI.

Models

Python classes representing DB tables.

Migrations

Track and apply DB schema changes.

Templates

Dynamic HTML with {{ vars }} and {% tags %}.

Creating Project/App

django-admin startproject / manage.py startapp.

manage.py

Command-line utility for project management.

Relations

ForeignKey, OneToOne, ManyToMany.

Static & Media Files

Static=assets, Media=user uploads.

■ Django - Intermediate Level

ORM

Abstracts database interaction via models.

QuerySets

Represent database queries, lazy evaluation.

Request/Response

Processed via middleware and views.

Middleware

Global request/response processing.

settings.py

Holds configuration (DB, middleware).

URL Dispatcher

Maps URLs to views via urls.py.

Signals

Notify when actions occur, e.g., post_save.

Form Validation

Use forms.Form or ModelForm with clean().

Auth System

Built-in user model, permissions, decorators.

select_related/prefetch_related

Optimize DB queries for related objects.

■ Django - Advanced Level

Caching Framework

Supports per-view/site/template caching.

Query Optimization

Use select_related, prefetch_related, pagination.

Transactions

Use transaction.atomic() for atomicity.

Deployment

Use Gunicorn + Nginx, Docker, WSGI/ASGI.

DRF

Framework for building REST APIs with serializers.

Pagination

Use PageNumberPagination in DRF.

Security

Enable CSRF, XSS protection, HTTPS.

Background Tasks

Use Celery or Django-RQ with Redis.

FBV vs CBV

FBV=simple, CBV=reusable, OOP-based.

Signals Internal

Implements observer pattern for event handling.