

Python Interview Questions for 2 Years Exp

Here are **top Python interview questions and answers** for someone with **2 years of experience**, covering **basic to intermediate** concepts typically asked in **real-world job interviews**:

◆ BASIC LEVEL QUESTIONS

1. What are Python's key features?

- Interpreted and dynamically typed
 - High-level language with simple syntax
 - Supports object-oriented and functional programming
 - Extensive standard library
 - Portable and platform-independent
 - Large community support
-

2. What is PEP 8?

PEP 8 is a **style guide** for writing Python code. It ensures code readability and consistency across Python projects.

3. What is the difference between a list and a tuple?

Feature	List	Tuple
Mutability	Mutable	Immutable

Syntax	[1, 2, 3]	(1, 2, 3)
Performance	Slower	Faster (due to immutability)
Use case	Dynamic data	Fixed/constant data

4. What is the difference between `is` and `==`?

- `==` compares **values**.
- `is` compares **object identity (memory address)**.

```
python
CopyEdit
a = [1, 2]
b = [1, 2]
print(a == b)  # True
print(a is b)  # False
```

5. What are Python decorators?

Decorators are functions that **modify the behavior of another function** without changing its code.

```
python
CopyEdit
def decorator(func):
    def wrapper():
        print("Before function")
        func()
        print("After function")
    return wrapper

@decorator
def greet():
    print("Hello")
```

◆ INTERMEDIATE QUESTIONS

6. Explain list comprehension with an example.

```
python
CopyEdit
squares = [x**2 for x in range(5)]
# Output: [0, 1, 4, 9, 16]
```

7. What is the difference between `deepcopy()` and `copy()`?

- `copy()` → Shallow copy (nested objects share reference).
- `deepcopy()` → Full independent clone.

```
python
CopyEdit
import copy

a = [[1, 2], [3, 4]]
b = copy.copy(a)
c = copy.deepcopy(a)
```

8. What are Python generators?

Generators are **functions that yield values one at a time** using the `yield` keyword. They are memory-efficient.

```
python
CopyEdit
def gen():
    yield 1
    yield 2

g = gen()
```

```
next(g) # 1
```

9. What is a lambda function?

Anonymous, single-expression function:

```
python
CopyEdit
add = lambda x, y: x + y
add(2, 3) # 5
```

10. Explain Python's Global Interpreter Lock (GIL).

The GIL is a mutex that **allows only one thread to execute at a time**, even on multi-core processors — mainly affects CPU-bound threads.

◆ OBJECT-ORIENTED QUESTIONS

11. What is the difference between `@classmethod`, `@staticmethod`, and instance methods?

- `@classmethod`: Accesses class-level data
 - `@staticmethod`: Utility function, doesn't access instance or class
 - Instance method: Regular method accessing instance (`self`)
-

12. What are `__init__` and `__str__`?

- `__init__`: Constructor
- `__str__`: Returns readable string representation

13. What is inheritance in Python?

Python supports **single**, **multiple**, **multilevel**, and **hierarchical** inheritance using classes.

```
python
CopyEdit
class A:
    pass

class B(A):
    pass # Inherits from A
```

◆ ERROR HANDLING

14. How do you handle exceptions in Python?

Using `try...except...finally`

```
python
CopyEdit
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print("Cleanup")
```

15. What is the use of **raise**?

Used to **manually throw exceptions**.

```
python
CopyEdit
if age < 0:
    raise ValueError("Age cannot be negative")
```

◆ ADVANCED OR PRACTICAL QUESTIONS

16. How to manage dependencies in Python?

Using `pip`, `venv`, and `requirements.txt`

bash

CopyEdit

```
pip freeze > requirements.txt
```

17. How do you read/write files in Python?

python

CopyEdit

```
with open('file.txt', 'r') as f:
    data = f.read()
```

```
with open('file.txt', 'w') as f:
    f.write("Hello")
```

18. What are Python modules and packages?

- **Module:** Single `.py` file
 - **Package:** Directory with `__init__.py` and multiple modules
-

19. Difference between `*args` and `**kwargs`?

python

CopyEdit

```
def example(*args, **kwargs):
    print(args)      # Tuple
    print(kwargs)    # Dict
```

20. What's your experience with any Python frameworks or libraries?

Be ready to talk about Django, Flask, Pandas, NumPy, etc., based on your work.

Would you like me to prepare **mock interview questions with answers tailored to your resume** or based on your recent work (like web dev, ML, etc.)?

You said:

top django interview questions and answers for 2 years experienced

ChatGPT said:

Here are the **top Django interview questions and answers** tailored for someone with **2 years of experience** — covering both **core concepts** and **practical application**.

◆ BASIC DJANGO INTERVIEW QUESTIONS

1. What is Django?

Django is a **high-level Python web framework** that encourages **rapid development** and **clean, pragmatic design**. It follows the **MTV architecture** (Model-Template-View).

2. What is MTV architecture in Django?

Component	Role
Model	Data structure (DB layer)
Template	UI layer (HTML + Django template tags)
View	Business logic (processes user request and returns response)

3. What are Django's main features?

- ORM (Object Relational Mapping)
 - Admin interface
 - Middleware support
 - Authentication system
 - Scalability and security features
 - Built-in development server
-

4. What is a Django project and Django app?

- **Project:** A full web application (settings, apps, URLs)
 - **App:** A modular component (like blog, users, orders) that can be reused across projects
-

5. How do you create a Django project and app?

bash

CopyEdit

```
django-admin startproject myproject  
python manage.py startapp myapp
```

◆ MODELS & ORM

6. How does Django ORM work?

Django ORM allows you to interact with the database using Python code rather than raw SQL.

python

CopyEdit

```
# models.py  
class Book(models.Model):  
    title = models.CharField(max_length=100)
```



```
# Query
Book.objects.all()
Book.objects.get(id=1)
```

7. What are model field types in Django?

Some common types:

- CharField
 - TextField
 - IntegerField
 - DateTimeField
 - BooleanField
 - ForeignKey, ManyToManyField, OneToOneField
-

8. What is makemigrations vs migrate?

Command	Description
<code>makemigrations</code>	Creates migration files
<code>migrate</code>	Applies changes to the database

9. How to filter records in Django ORM?

```
python
CopyEdit
Book.objects.filter(author='John')
```

```
Book.objects.exclude(price__lt=100)
Book.objects.order_by('-published_date')
```

10. What is a QuerySet?

A lazy, iterable object that represents a collection of database records.

◆ VIEWS & URLS

11. Difference between function-based and class-based views?

- **Function-based views (FBV):** Simple logic using Python functions.
- **Class-based views (CBV):** Reusable logic with inheritance and mixins.

```
python
CopyEdit
# FBV
def home(request):
    return render(request, 'home.html')

# CBV
from django.views import View
class HomeView(View):
    def get(self, request):
        return render(request, 'home.html')
```

12. How do you configure URLs in Django?

```
python
CopyEdit
# myapp/urls.py
from django.urls import path
from . import views
```

```
urlpatterns = [  
    path('', views.home, name='home')  
]  
  
# project/urls.py  
path('myapp/', include('myapp.urls'))
```

◆ TEMPLATES

13. What are Django template tags?

Used to render dynamic content in HTML.

```
html  
CopyEdit  
<h1>{{ book.title }}</h1>  
{% if user.is_authenticated %}  
    <p>Welcome, {{ user.username }}</p>  
{% endif %}
```

14. How to include static files and media in Django?

In **settings.py**:

```
python  
CopyEdit  
STATIC_URL = '/static/'  
MEDIA_URL = '/media/'
```

In **template**:

```
html  
CopyEdit  
{% load static %}  

```

◆ FORMS

15. Difference between `ModelForm` and regular `Form`?

- `Form`: Manually define fields.
- `ModelForm`: Auto-generates fields from model.

```
python
CopyEdit
class BookForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = ['title', 'author']
```

16. How do you handle form validation in Django?

Using `is_valid()` and `clean()` methods inside form classes.

◆ DJANGO ADMIN

17. How do you customize the Django admin interface?

```
python
CopyEdit
from django.contrib import admin
from .models import Book

class BookAdmin(admin.ModelAdmin):
    list_display = ['title', 'author']

admin.site.register(Book, BookAdmin)
```

◆ MIDDLEWARE, AUTH & SECURITY

18. What is middleware in Django?

Middleware is a hook to process requests/responses globally before they reach views or after the response leaves the view.

Example: `AuthenticationMiddleware`, `SessionMiddleware`.

19. How does authentication work in Django?

Django provides a built-in `User` model, login/logout views, and session-based authentication.

```
python
CopyEdit
from django.contrib.auth import authenticate, login

user = authenticate(username='john', password='secret')
if user:
    login(request, user)
```

20. How do you protect against CSRF in Django?

Django uses the `{% csrf_token %}` template tag and middleware to prevent Cross-Site Request Forgery.

◆ DEPLOYMENT & BEST PRACTICES

21. How do you deploy a Django project?

- Use **Gunicorn** or **uWSGI**
- Use **Nginx/Apache** as a reverse proxy
- Setup **PostgreSQL** for production DB
- Use **environment variables** for secrets

- Use `DEBUG=False` in production
-

22. How to handle static files in production?

Use `collectstatic` to gather all static files into one folder.

bash

CopyEdit

```
python manage.py collectstatic
```

23. What are common security best practices in Django?

- Use `DEBUG=False` in production
 - Use `ALLOWED_HOSTS`
 - Use HTTPS (via `SECURE_SSL_REDIRECT`)
 - Store secrets in environment variables
 - Enable CSRF and XSS protection
-

24. How to create API in Django?

Using **Django REST Framework (DRF)**:

python

CopyEdit

```
from rest_framework.views import APIView
from rest_framework.response import Response
```

```
class HelloView(APIView):
    def get(self, request):
        return Response({"message": "Hello, world!"})
```

◆ BEHAVIORAL/REAL-WORLD QUESTIONS

25. What Django projects have you worked on?

Be prepared to talk about:

- The project structure
- Custom models/forms
- User authentication
- Admin customization
- API endpoints
- Deployment challenges

Would you like mock interview questions with scenario-based tasks or questions based on your specific Django projects?

You said:

top flask interview questions and answers for 2 years experienced

ChatGPT said:

Here are the **Top Flask Interview Questions and Answers** tailored for someone with **2 years of experience**, covering **basics**, **internals**, and **real-world scenarios**:

◆ BASIC FLASK INTERVIEW QUESTIONS

1. What is Flask?

Flask is a **lightweight, micro web framework** in Python used to build web applications quickly. It is minimal but extensible and follows the **WSGI** standard.

2. What are Flask's key features?

- Lightweight and modular
 - Built-in development server and debugger
 - RESTful request handling
 - Jinja2 templating engine
 - Extension support (Flask-SQLAlchemy, Flask-Login, etc.)
 - No ORM or form validation by default (can be added as needed)
-

3. How do you create a basic Flask app?

python

CopyEdit

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return 'Hello, Flask!'

if __name__ == '__main__':
    app.run(debug=True)
```

4. What is the use of `app.run(debug=True)`?

- `debug=True` enables:
 - Debugger (for interactive error pages)
 - Auto-reload when code changes
 - Detailed error logs

5. How do you define routes in Flask?

Using the `@app.route()` decorator.

```
python
CopyEdit
@app.route('/about')
def about():
    return "About Page"
```

◆ FLASK TEMPLATES & FORMS

6. What is Jinja2 in Flask?

Flask uses **Jinja2** as its templating engine to generate HTML dynamically.

Example:

```
html
CopyEdit
<h1>Hello, {{ name }}</h1>
{% if logged_in %}
    <p>Welcome back!</p>
{% endif %}
```

7. How to pass data to a template?

```
python
CopyEdit
from flask import render_template

@app.route('/profile')
def profile():
    return render_template('profile.html', name="John")
```

8. How do you handle forms in Flask?

- Using `request.form` for basic form handling
- Or use `Flask-WTF` for secure and structured form handling

```
python
CopyEdit
from flask import request

@app.route('/submit', methods=['POST'])
def submit():
    name = request.form['name']
    return f"Hello, {name}"
```

◆ FLASK ROUTING & REQUESTS

9. How do you handle different HTTP methods in Flask?

```
python
CopyEdit
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return 'Logged in!'
    return 'Login Form'
```

10. What is `url_for()` in Flask?

It dynamically generates URLs from function names.

```
python
CopyEdit
url_for('home') # Returns "/"
```

11. How do you redirect in Flask?

```
python
CopyEdit
from flask import redirect, url_for

@app.route('/go-home')
def go_home():
    return redirect(url_for('home'))
```

◆ FLASK SESSIONS & COOKIES

12. What is Flask **session** and how does it work?

Flask uses a **secure cookie-based session** to store user data between requests.

```
python
CopyEdit
from flask import session

@app.route('/set/')
def set_session():
    session['user'] = 'John'
    return 'Session Set'
```

You need to set a **SECRET_KEY** in your app config to use sessions.

13. How to use cookies in Flask?

```
python
CopyEdit
from flask import make_response, request

@app.route('/setcookie')
def setcookie():
    resp = make_response("Cookie Set")
    resp.set_cookie('username', 'John')
```

```
return resp
```

◆ FLASK DATABASE & ORM

14. Which ORMs are used with Flask?

- Flask-SQLAlchemy (most popular)
 - Flask-Peewee
 - Flask-MongoEngine (for MongoDB)
-

15. How do you connect Flask to a database using SQLAlchemy?

python

CopyEdit

```
from flask_sqlalchemy import SQLAlchemy
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
```

```
db = SQLAlchemy(app)
```

```
class User(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    username = db.Column(db.String(80), unique=True, nullable=False)
```

16. How do you run migrations in Flask?

Using **Flask-Migrate**:

bash

CopyEdit

```
flask db init
```

```
flask db migrate -m "initial"
```

```
flask db upgrade
```

◆ FLASK BLUEPRINTS & STRUCTURE

17. What are Blueprints in Flask?

Blueprints help you **organize a large Flask app** into smaller modules.

```
python
CopyEdit
from flask import Blueprint

admin = Blueprint('admin', __name__)

@admin.route('/dashboard')
def dashboard():
    return "Admin Dashboard"
```

Then register in main app:

```
python
CopyEdit
app.register_blueprint(admin, url_prefix='/admin')
```

18. What is Flask's application factory pattern?

A design pattern to create and configure the app in a function instead of a global object. Useful for testing and configuration.

```
python
CopyEdit
def create_app():
    app = Flask(__name__)
    app.config.from_object('config')
    return app
```

◆ FLASK SECURITY & AUTHENTICATION

19. How do you implement login in Flask?

Using **Flask-Login**:

```
python
CopyEdit
from flask_login import LoginManager

login_manager = LoginManager()
login_manager.init_app(app)
```

It handles session, login state, and decorators like `@login_required`.

20. How to prevent CSRF in Flask forms?

Use **Flask-WTF**, which has built-in CSRF protection via `{{ form.csrf_token }}`.

◆ PRACTICAL & DEPLOYMENT QUESTIONS

21. How do you deploy a Flask app?

- Use **Gunicorn** as WSGI server
 - Use **Nginx/Apache** as reverse proxy
 - Use **Docker** for containerized deployment
 - Set `debug=False` in production
-

22. What are common Flask extensions you've used?

- Flask-SQLAlchemy (DB)
- Flask-Migrate (migrations)
- Flask-WTF (forms)

- Flask-Login (auth)
 - Flask-Mail (emails)
 - Flask-RESTful or Flask-API (API support)
-

23. How do you handle errors in Flask?

python

CopyEdit

```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404
```

24. How do you structure a scalable Flask application?

- Use Blueprints
 - Application factory
 - Separate config files (dev, prod, test)
 - Modularized code (models, views, forms, etc.)
-

25. How do you create a REST API with Flask?

python

CopyEdit

```
from flask import jsonify

@app.route('/api/data')
def api_data():
    return jsonify({"message": "Hello, API"})
```

Or use **Flask-RESTful** for advanced support.

Would you like **mock questions** or **real project-based scenarios** you can practice for a Flask interview? I can also help with **system design questions using Flask** if needed.

You said:

top fastapi interview questions and answers for 2 years experienced

ChatGPT said:

Here are the **Top FastAPI Interview Questions and Answers** tailored for someone with **2 years of experience**, especially for backend developers working on modern APIs with **Python**.

◆ BASICS OF FASTAPI

1. What is FastAPI?

FastAPI is a **modern, high-performance web framework** for building APIs with **Python 3.7+** based on **standard Python type hints**.

- Built on **Starlette** (web microframework) and **Pydantic** (for data validation)
- Auto-generates **OpenAPI (Swagger)** docs
- Asynchronous support using **async/await**
- Fast — on par with Node.js and Go in benchmarks

2. Why choose FastAPI over Flask or Django?

Feature	FastAPI	Flask	Django
Async Support	✓ Built-in	✗ Not native	✗ Complex
Type Safety	✓ Yes (Pydantic)	✗ No	✗ No
Auto Docs	✓ Swagger	✗ Manual	✗ Add-ons
Performance	🔥 Very High	Moderate	Moderate

3. How do you create a basic FastAPI app?

```
python
CopyEdit
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello, FastAPI!"}
```

Run with:

```
bash
CopyEdit
uvicorn main:app --reload
```

◆ ROUTING & HTTP METHODS

4. How do you define different HTTP methods in FastAPI?

```
python
CopyEdit
@app.get("/items")
def get_items(): ...

@app.post("/items")
def create_item(): ...
```

Supports: GET, POST, PUT, DELETE, PATCH

5. How do you use path parameters and query parameters?

```
python
CopyEdit
@app.get("/items/{item_id}")
```

```
def read_item(item_id: int, q: str = None):  
    return {"item_id": item_id, "q": q}
```

6. What is **Path** and **Query** in FastAPI?

Used for validating parameters:

```
python  
CopyEdit  
from fastapi import Path, Query  
  
@app.get("/books/{book_id}")  
def get_book(book_id: int = Path(..., gt=0), q: str = Query(None,  
max_length=50)):  
    ...
```

◆ REQUEST & RESPONSE HANDLING

7. How do you handle request bodies in FastAPI?

Use **Pydantic models**.

```
python  
CopyEdit  
from pydantic import BaseModel  
  
class Item(BaseModel):  
    name: str  
    price: float  
  
@app.post("/items/")  
def create_item(item: Item):  
    return item
```

8. How do you return custom responses in FastAPI?

```
python
CopyEdit
from fastapi.responses import JSONResponse

@app.get("/custom")
def custom_response():
    return JSONResponse(content={"message": "Custom"},
status_code=202)
```

9. How do you add response models (output validation)?

```
python
CopyEdit
@app.post("/items/", response_model=Item)
def create_item(item: Item):
    return item
```

◆ ASYNC AND PERFORMANCE

10. Is FastAPI asynchronous? How do you use **async**?

Yes. You can write async routes like:

```
python
CopyEdit
@app.get("/async-task")
async def do_async():
    await some_async_func()
    return {"done": True}
```

Async boosts performance during I/O-bound operations (e.g., DB calls, HTTP requests).

◆ VALIDATION & Pydantic

11. What is Pydantic and why is it used in FastAPI?

Pydantic is used for:

- Data parsing
- Validation
- Type enforcement

Example:

```
python
CopyEdit
class User(BaseModel):
    name: str
    age: int = Field(..., gt=0)
```

12. How to add custom validation in Pydantic?

```
python
CopyEdit
from pydantic import validator

class Item(BaseModel):
    name: str

    @validator('name')
    def name_must_be_alpha(cls, v):
        if not v.isalpha():
            raise ValueError("Must be alphabetic")
        return v
```

◆ FASTAPI FEATURES

13. How do you document your API in FastAPI?

FastAPI auto-generates **Swagger UI** and **ReDoc** at:

- `/docs`
 - `/redoc`
-

14. What is `Depends()` in FastAPI?

Used for **dependency injection** — like auth, DB connection, etc.

```
python
CopyEdit
from fastapi import Depends

def get_db():
    return db_session

@app.get("/users/")
def read_users(db=Depends(get_db)):
    ...
```

15. How do you implement middleware in FastAPI?

```
python
CopyEdit
@app.middleware("http")
async def log_requests(request, call_next):
    response = await call_next(request)
    return response
```

◆ SECURITY & AUTHENTICATION

16. How do you handle authentication in FastAPI?

Using `fastapi.security`:

```
python
```

CopyEdit

```
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/users/me")
def read_users_me(token: str = Depends(oauth2_scheme)):
    return {"token": token}
```

17. What are scopes and roles in OAuth2 with FastAPI?

Scopes define what actions a token is authorized to perform. FastAPI supports **scope-based authorization** via dependencies.

◆ DATABASE & BACKEND INTEGRATION

18. How to connect a database in FastAPI?

Usually with **SQLAlchemy**, **Tortoise ORM**, or **MongoDB (Motor)**.

Example (SQLAlchemy):

python

CopyEdit

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

engine = create_engine("sqlite:///./test.db")
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
                             bind=engine)
```

Inject using `Depends`.

19. How do you handle background tasks?

python

CopyEdit

```
from fastapi import BackgroundTasks

def send_email(email: str):
    # send email logic

@app.post("/send/")
def send(email: str, background_tasks: BackgroundTasks):
    background_tasks.add_task(send_email, email)
    return {"message": "Will send email soon"}
```

20. How to handle CORS in FastAPI?

python

CopyEdit

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Or specific origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

◆ DEPLOYMENT & TESTING

21. How do you deploy FastAPI?

Use **uvicorn** or **gunicorn** with workers:

bash

CopyEdit

```
uvicorn main:app --host 0.0.0.0 --port 8000 --workers 4
```

Behind **Ngixn** or **Docker** in production.

22. How do you write unit tests for FastAPI?

python

CopyEdit

```
from fastapi.testclient import TestClient
```

```
client = TestClient(app)
```

```
def test_home():  
    response = client.get("/")  
    assert response.status_code == 200
```

23. What are FastAPI's limitations?

- Still maturing compared to Django
- Async ORM options are evolving
- Requires familiarity with type hints and async

24. What are some common FastAPI extensions you've used?

- FastAPI-Users (auth)
 - Tortoise ORM
 - SQLAlchemy
 - Alembic (migrations)
 - Pydantic
 - Celery (background jobs)
-

25. How do you structure a FastAPI project for scalability?

bash

CopyEdit

/app

 /routers

 /models

 /schemas

 /services

 /core (config, auth)

main.py

Use modular routers, dependency injection, and environment-based settings.