1. **Why dict takes only immutable datatypes as keys?**

Answer Keywords – Hashing, unique key, hashability, immutable cannot be changed after declaration

1. **Other than list, which other datatypes supports comprehension technique?**

Answer Keywords – Dict Comprehension and Set Comprehension

**3. What is the difference between “double equal to” (==) and “is” operator?**

Answer Keywords - == checks exact value, “is” checks for same object or memory location based

**4. What is the time complexity or Big O (pronounced as Big Ohh) notation of list.sort, list.append, list.index**

Ans – sort is N(Log N), append is O(1) or constant time, index is O(n) or linear

**5. Why is set unordered?**

Ans keywords – Items are hashed, hashable, allows only immutable types

**6. What is context manager?**

Ans – “with clause”. Context manager ensures graceful opening/entry and closing/exit, and maintains the context.

**8. What is MRO, and when MRO plays role?**

Ans keywords - Method Resolution Order. MRO plays role when there is multiple inheritance.

**9. Does Python supports operator overloading?**

Ans – Yes. Magic methods or Dunder methods

**10. What is a singleton class? How to implement singleton class in python?**

Ans Keywords – Only one instance of class. Single object or one object. Modify \_\_new\_\_ magic method to implement.

**11. What is blueprint in Flask?**

Ans keywords – Organize APIs, common patterns, register routes, url prefix, sub app

**12. What do you mean by stateless with respect to REST APIs?**

Ans keywords – Independent requests, no sessions, no need to track multiple communication, no client server dependency

**13. What are Flask middlewares?**

Ans keywords – before\_request, after\_request, common methods that gets executed before or after each request.

**14. What is the difference between one, first and scalar calls in SqlAlchemy?**

Ans keywords – One – Expects one and only one record. Raises exception if record not found or multiple records found. Scalar – Similar to One call, but raise no exception if record not found. First – Returns first row out of multiple filtered row

**FastAPI Questions:**

1. **What are the key features of FastAPI that make it popular for building APIs?**

**Ans:** FastAPI is known for being fast, asynchronous, and easy to use. Key features include automatic validation via Pydantic, support for asynchronous programming, automatic interactive API documentation (OpenAPI), and great performance, comparable to frameworks like Node.js and Go.

1. **How does dependency injection work in FastAPI?**

**Ans:** FastAPI’s dependency injection system allows you to define reusable components that can be injected into endpoints. This is done by defining a function with dependencies (such as database connections or external services), and using the Depends() function to pass them to routes.

1. **How does FastAPI handle async and await?**

**Ans:** FastAPI fully supports asynchronous programming using async and await. Endpoints defined with async can perform non-blocking I/O operations, which allows the application to serve many requests concurrently, improving scalability and performance.

1. **How does FastAPI handle validation, and what role does Pydantic play?**

**Ans:** FastAPI uses Pydantic for data validation. It automatically validates request bodies, query parameters, and path parameters by defining Pydantic models, which enforce type hints and provide automatic data validation and error reporting. This is one of the core strengths of FastAPI.

1. **How would you implement OAuth2 authentication in FastAPI?**

**Ans:** FastAPI provides built-in support for OAuth2 authentication, including password flow. You can use the OAuth2PasswordBearer class to define the authentication scheme, and create a token endpoint that returns a JWT token. This token is then validated on protected endpoints via dependency injection.

1. How can you handle CORS (Cross-Origin Resource Sharing) in a FastAPI application?

**Ans:** FastAPI has built-in support for handling CORS using the fastapi.middleware.cors.CORSMiddleware class. You can specify which domains, methods, and headers are allowed for cross-origin requests by adding this middleware to your application.

1. **How do background tasks work in FastAPI?**

**Ans:** FastAPI provides a BackgroundTasks class for running tasks in the background after returning a response. This is useful for tasks that don’t need to be completed while the user waits, such as sending an email or processing a file. The task is scheduled after the response is sent to the client.

1. How do you handle file uploads in FastAPI?

**Ans:** FastAPI makes handling file uploads straightforward with the File and UploadFile types. The UploadFile type is preferred because it doesn’t load the file into memory and allows efficient streaming of large files. Files can be accessed by declaring them as dependencies in the route.

**Flask Questions:**

1. **What are the main differences between Flask and FastAPI?**

**Ans:** Flask is a more traditional web framework with synchronous request handling, while FastAPI is asynchronous, providing better performance for I/O-bound operations. FastAPI offers built-in validation and type hints, while Flask requires third-party libraries for many features.

1. How do you implement middleware in Flask?

**Ans:** Middleware can be implemented in Flask by creating a WSGI middleware component or by using Flask's before\_request and after\_request hooks. These hooks allow code to run before and after each request.

1. What are Python decorators, and how are they used?

**Ans:** Decorators are a way to modify the behavior of a function or class method. They are used to wrap another function in order to extend or alter its behavior, typically used for logging, authentication, or access control. The @ symbol is used to apply a decorator to a function.

1. How does Python’s memory management work?

**Ans:** Python uses reference counting and a garbage collector for memory management. When an object's reference count drops to zero, the object is destroyed. Python's garbage collector handles circular references via a generational approach.

1. Can you explain the difference between shallow copy and deep copy in Python?

**Ans:** A shallow copy creates a new object but inserts references into the same elements as the original. A deep copy creates a new object and recursively copies all objects found in the original, so the new object is completely independent of the old one.

1. How does the Global Interpreter Lock (GIL) affect multi-threading in Python?

**Ans:** The GIL is a mutex that prevents multiple native threads from executing Python bytecodes at once. It can be a performance bottleneck in CPU-bound tasks as it limits multi-threaded execution. For I/O-bound tasks, it’s less of an issue as threads can be released while waiting for I/O operations.

1. What are Python’s context managers, and how are they implemented?

**Ans:** Context managers are used to properly manage resources (e.g., files, database connections). They are implemented using the with statement, which ensures resources are properly acquired and released. A custom context manager can be created by implementing \_\_enter\_\_ and \_\_exit\_\_ methods or using the contextlib module’s @contextmanager decorator.

1. How do you handle exceptions in Python, and what is the finally clause used for?

**Ans:** Exceptions in Python are handled using try, except, and finally blocks. The finally clause is used to execute code no matter whether an exception occurs or not, usually for cleanup tasks like closing files or network connections.

1. Can you explain what Python’s asyncio library is and how it differs from traditional multi-threading?

**Ans:** asyncio is used for writing concurrent code using async and await for cooperative multitasking. Unlike multi-threading, which pre-empts threads, asyncio allows multiple tasks to be executed in a single thread by yielding control during I/O-bound tasks. This makes asyncio ideal for I/O-bound applications like web servers.

1. What are metaclasses in Python, and when would you use one?

**Ans:** Metaclasses are the classes of classes, meaning they define the behavior of class objects. They can be used to control class creation, such as adding or modifying methods and attributes, or enforcing certain constraints during the class construction process.