PYTHON 3, DOCKER:

1. Explain the difference between deep copy and shallow copy in Python. How do you implement each?

A **shallow copy** (using copy.copy()) creates a new object but does not clone the nested objects within it. Instead, it maintains references to the original inner objects. This means any changes made to those nested elements in the copied object will also be reflected in the original. Shallow copying is useful when you only need a new outer structure but don't mind sharing inner data.

In contrast, a **deep copy** (using copy.deepcopy()) creates a completely independent replica of the original object, including all nested structures. No shared references exist, so modifying the copy won’t affect the original. This is ideal for duplicating complex, mutable data safely.

2. What are decorators in Python? Can you write a custom decorator that shows times the execution of a function?

Decorators in Python are a way to change or add something to a function without touching its original code. You can think of them as functions that take up another function and give back a new one with some extra work added. They are useful when you want to reuse the same extra feature (like checking time or logging) with many functions, without writing the same code again and again.

3. How does Python's garbage collection (GC) work, and how can it affect performance?

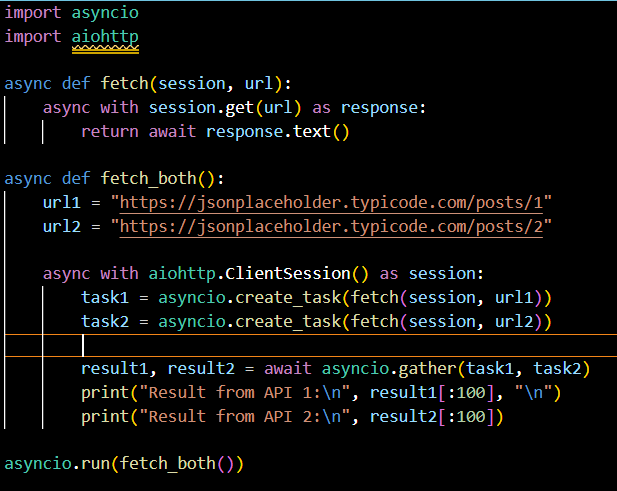
Python’s **garbage collection (GC)** automatically manages memory by cleaning up objects that are no longer used. It mainly works using **reference counting**—each object keeps track of how many references point to it. When this count becomes zero, the memory is freed.

However, reference counting alone can’t handle **circular references** (when two objects refer to each other), so Python also uses a **cyclic garbage collector** to detect and clean these cycles.

GC can affect performance because:

* It pauses your program briefly during collection.
* Frequent creation/deletion of objects may trigger GC more often.

4. Write a Python function that fetches data from two APIs simultaneously using asyncio and await.



A black screen with colorful text

AI-generated content may be incorrect.

5. What is the GIL (Global Interpreter Lock) in Python, and how does it affect concurrency with async programming?

The Global Interpreter Lock (GIL) in Python allows only one thread to execute Python bytecode at a time, limiting true parallelism in multi-threading. However, GIL doesn’t affect asynchronous programming (async/await), since async tasks run in a single thread, switching efficiently during I/O operations without needing multiple threads.

6. How do you persist data in Docker containers? Explain the use of volumes and bind mounts.

In Docker, data inside containers is temporary and lost when the container stops. To persist data, we use **volumes** or **bind mounts**. **Volumes** are managed by Docker and stored in a special location on the host. They are ideal for sharing data between containers and for backup. **Bind mounts, map** a specific path on the host machine to a path in the container, allowing real-time access to host files. While bind mounts give more control, they are less portable. Volumes are preferred for consistent data management in production. Both methods help retain data across container restarts or rebuilds.

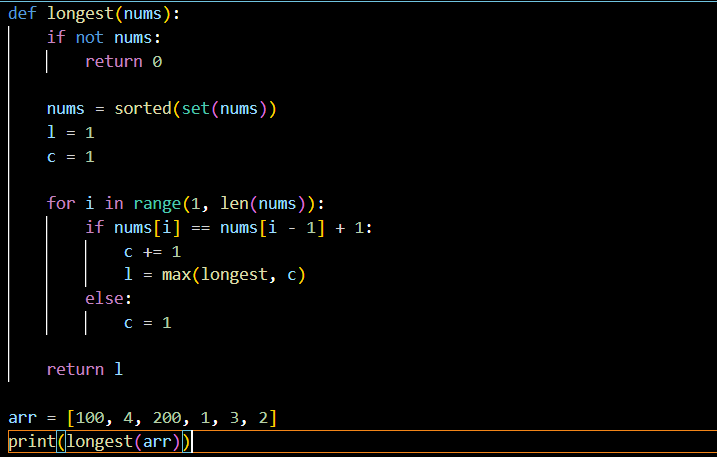
7. Given an unsorted array of integers, you need to write a program to find the length of the longest

consecutive elements sequence.

For example,

Input: [100, 4, 200, 1, 3, 2]

Output: 4 # The sequence [1, 2, 3, 4] is the longest consecutive sequence.



FRAMEWORKS:

1. What is the difference between ForeignKey, OneToOneField, and ManyToManyField in Django models? Provide examples where each might be used.

I don’t know much about the Django framework, but on looking it up on GeeksforGeeks, here’s what I understood:

In Django models:

* **ForeignKey** is used for a **many-to-one** relationship. For example, if many blog posts belong to one author:

A screen shot of a computer code

AI-generated content may be incorrect.

* **OneToOneField** means one object is related to exactly one other object. For example, each user can have only one profile:

A screen shot of a computer code

AI-generated content may be incorrect.

* **ManyToManyField** is used when multiple records can relate to multiple others. Like a student enrolled in many courses, and each course had many students:

A computer code with white text

AI-generated content may be incorrect.

2. How would you optimize a Django ORM query to improve performance?

We can optimize Django ORM queries by using select\_related() and prefetch\_related() to reduce database hits. We can also use .only() or .defer() to fetch only required fields. Adding db\_index=True improves lookup speed. Additionally, using annotate() for aggregations and pagination for large datasets helps improve performance.

3. Explain Django middleware. Can you write a custom middleware that logs request duration

Django **middleware** is a lightweight plugin that processes **requests and responses** globally. It sits between the Django request/response cycle and can be used for tasks like logging, authentication, modifying headers, or handling exceptions.



4. What are Django signals, and how would you use them in a real-world project?

Django signals are used to perform actions automatically when certain events occur in the application, like after saving or deleting a model. We can use them to send welcome emails after user registration, log user activities, create related profiles, or clean up resources when objects are deleted.

5. Can you provide an example of asynchronous route handling in FastAPI?

A computer screen with text

AI-generated content may be incorrect.

This FastAPI route handles requests asynchronously by waiting 2 seconds using asyncio.sleep before returning a response, allowing other requests to be processed in the meantime.

6. Why is FastAPI generally considered faster than Flask?

FastAPI is generally considered faster than Flask because:

* **Asynchronous Support**: Built-in support for async/await enables non-blocking I/O and high concurrency.
* **Automatic Validation**: Uses Pydantic for fast data parsing and validation based on Python type hints.
* **Fewer Context Switches**: ASGI architecture handles multiple requests more efficiently than Flask's WSGI.
* **Modern Features**: Designed with modern Python features like type hints and async from the ground up, unlike Flask which is synchronous by default.

PANDAS & NUMPY:

1. How would you handle missing data in a Pandas DataFrame? Describe at least three strategies.

We need to handle missing data in a DataFrame because there might be some feature which can be lost if we remove rows with missing values. We can use:

1. **Remove Missing Data**: Use dropna() to remove rows or columns with missing values, useful when the missing data is minimal and won’t affect analysis.
2. **Fill with Default Values**: Use fillna() to replace missing values with a constant (like 0 or "Unknown") or a statistical value like mean, median, or mode.
3. **Forward or Backward Fill**: Use ffill() or bfill() to propagate the previous or next non-missing value, useful in time series data to maintain continuity.

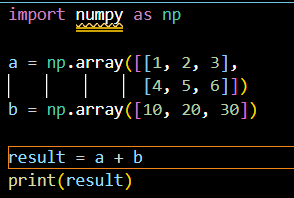
2. How would you optimize a large Pandas DataFrame for performance and memory usage?

To optimize a large Pandas DataFrame for performance and memory usage, we can use the following strategies:

1. **Use appropriate data types**: Convert columns to smaller types (e.g., int64 to int8, float64 to float32, object to category) using astype().
2. **Drop unnecessary columns**: Remove columns not needed for analysis to reduce memory.
3. **Use vectorized operations**: Avoid loops; use Pandas’ built-in functions which are optimized in C.
4. **Use inplace=True**: Modify data without creating copies when possible.
5. **Use Parquet files**: Store and read data using .parquet format, which is more efficient and compressed compared to CSV.

3. Can you explain broadcasting in Numpy and provide a practical example?

I don't know much about broadcasting in NumPy, but from what I read, it's a way that NumPy automatically makes arrays with different shapes work together during operations. Instead of manually resizing arrays, NumPy "stretches" smaller ones so operations like addition or multiplication can happen without errors.



Output:  
[[11 22 33]

[14 25 36]]

DATABASE:

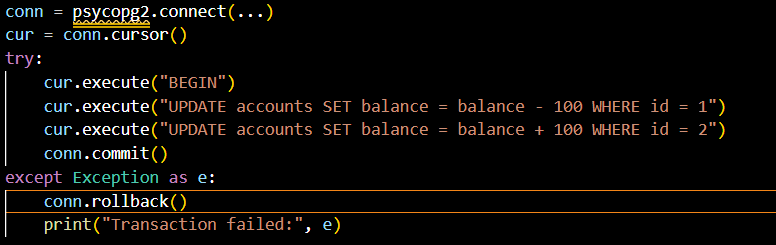
1. What is the difference between an INNER JOIN and a LEFT JOIN in SQL?

An **INNER JOIN** returns only the matching rows from both tables based on a common key. A **LEFT JOIN** returns all rows from the left table, and matching rows from the right table; if no match exists, NULLs are returned for the right table’s columns.

2. Explain how transactions work in PostgreSQL. How would you implement a rollback mechanism?

In PostgreSQL, a **transaction** is a sequence of operations executed as a single unit. It ensures **ACID** properties (Atomicity, Consistency, Isolation, Durability). A transaction starts with BEGIN, changes data with SQL statements, and ends with COMMIT to save or ROLLBACK to undo changes if something goes wrong.

If an error occurs, use ROLLBACK to undo all changes made after BEGIN. In Python (using psycopg2 or similar libraries), this is typically handled with try-except:



3. How would you use Django ORM to run raw SQL queries in PostgreSQL? Provide an example.

A screen shot of a video

AI-generated content may be incorrect.

4. Explain the benefits of using JSON fields in PostgreSQL.

Using JSON fields in PostgreSQL allows flexible, schema-less storage of structured data within a relational database. It’s useful for storing dynamic or nested data without altering table schemas. JSON functions and indexing enable efficient querying and filtering, combining relational power with NoSQL-like flexibility in a single system.

5. What is normalization? Explain 3NF

**Normalization** is the process used to reduce redundancy and improve data integrity. It breaks large tables into smaller ones and defines relationships between them.

**Third Normal Form (3NF)** is achieved when:

1. The table is in **Second Normal Form (2NF)**.
2. **No transitive dependency** exists—i.e., non-key columns depend only on the primary key, not on other non-key columns.

This ensures data is clean, avoids duplication, and makes updates more efficient.

6. What is database sharding?

I don't know much about database sharding, but from what I read, it's a way to split a large database into smaller, faster parts called **shards**. Each shard holds a portion of the data and runs on a separate server. This helps improve performance and scalability, especially for big applications.