Task 1: Describe in your own words what is GIL in python, and the pros and cons of it.

The Global Interpreter Lock (GIL) in Python is a mutual exclusion lock that restricts access to Python objects, allowing only one thread to execute Python bytecode at a time. Consequently, in a multi-threaded Python program, only a single thread can run Python code simultaneously.

Some of the advantages of using GIL includes:

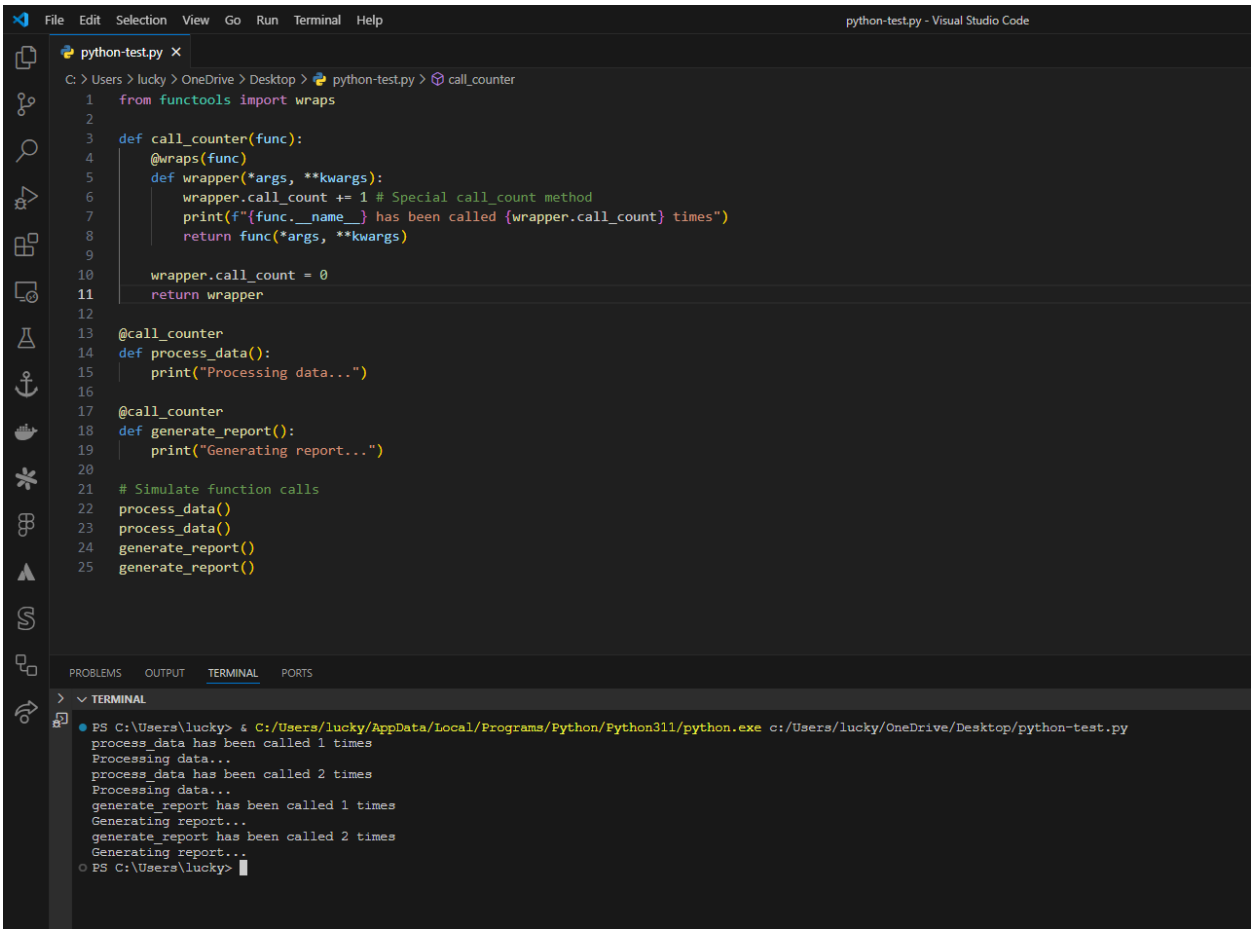
- **Decreased Danger of Racing Conditions**
  The GIL makes thread-safe programming easier for developers by lowering the possibility of race situations when multiple threads interact with Python objects.
- **Efficacy in Application**:
  The GIL simplifies the implementation of CPython, which is the Python reference implementation. It manages objects and memory without requiring complicated locking mechanisms.
- **Enhancement of Single-threaded Programs' Performance:**
  The GIL introduces very little overhead in single-threaded programs, and performance can be enhanced by making certain modifications.

Some of the disadvantages of using GIL includes:

- **Limitations on Multi-threading Performance**:
  The GIL can be a severe bottleneck in CPU-bound multi-threaded programs by preventing many threads from processing Python bytecode concurrently. This inhibits the capacity to effectively use multi-core processors.

- **Inefficient Multi-Core Systems:**
  In a multi-core system, the GIL can cause inefficiencies by underutilizing many cores while threads wait for the GIL rather than executing in parallel.

Task 2: Write a decorator in python that will count how many times the decorated function was called. It should print the number every time the decorated function is executed. Each function should be counted separately.



Link:

https://github.com/PiusLucky/Ergeon-Senior-Fullstack-Interview/blob/main/Tasks/task2.py

Task 3: If you see that a SQL SELECT query is slow - what would you do to improve it?

If a SQL `SELECT` query is running slow, there are several strategies to improve its performance. Here's a detailed approach:

1. Index Optimization:

- **Add Indexes:** Ensure that columns used in `WHERE`, `JOIN`, `ORDER BY`, and `GROUP BY` clauses are indexed.
- **Use Composite Indexes:** If multiple columns are frequently used together in queries, a composite index might be beneficial.
- **Remove Unused Indexes:** Over-indexing can slow down write operations and increase maintenance overhead.

2. Query Refactoring:

- **Select Only Necessary Columns:** Avoid `SELECT *`; specify only the columns you need.
- **Optimize Joins:** Ensure joins are using indexed columns and consider the order of joins.

3. Caching:

- **Use Query Caching:** If the data does not change frequently, caching the results of frequent queries can improve performance.
- **Application-level Caching:** Use tools like Redis or Memcached to cache query results at the application level

Task 4: What are the differences between "arrow" and "traditional" functions in JavaScript?

• Syntax**:**

- **Traditional Function:** Defined with the `function` keyword.
- **Arrow Function:** Uses the `=>` syntax, offering a more concise way to write functions.

• this **Binding:**

- **Traditional Function:** The `this` value is dynamic and depends on how the function is called.
- **Arrow Function:** The `this` value is lexically bound, meaning it uses the `this` value from the surrounding context where the function was defined.

• arguments **Object:**

- **Traditional Function:** Has its own `arguments` object, which contains all the arguments passed to the function.
- **Arrow Function:** Does not have its own `arguments` object. Instead, rest parameters (`...args`) must be used to access arguments.

• new **Keyword:**

- **Traditional Function:** Can be used as a constructor with the `new` keyword to create instances.
- **Arrow Function:** Cannot be used as a constructor and will throw an error if used with `new`.

• Implicit **Return:**

- **Traditional Function:** Requires an explicit `return` statement to return a value.
- **Arrow Function:** Can implicitly return a value if the function body is a single expression without braces.

• Methods**:**

- **Traditional Function:** Suitable for defining methods within objects due to its dynamic `this` binding.
- **Arrow Function:** Not suitable for methods that need their own `this` context, as `this` is lexically bound.

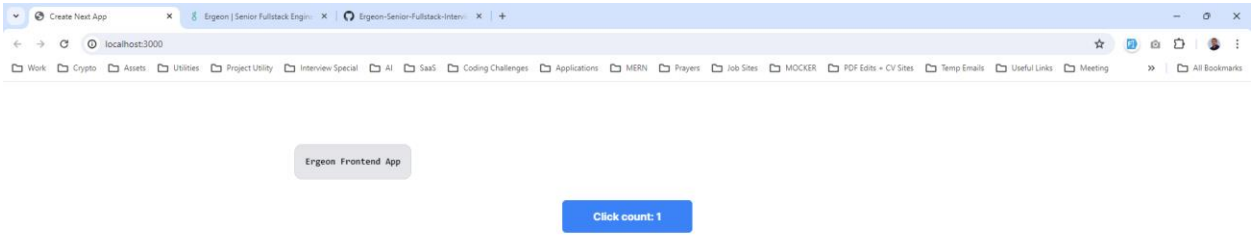
Link:

https://github.com/PiusLucky/Ergeon-Senior-Fullstack-Interview/blob/main/Tasks/task4.js

Task 5: Write a basic React component showing number of clicks on its button, use images below as example, allow to configure initial value of click count.

Link:

https://github.com/PiusLucky/Ergeon-Senior-Fullstack-Interview/tree/main/Tasks/task5



Task 6: Django Queries

Link:

https://github.com/PiusLucky/Ergeon-Senior-Fullstack-Interview/blob/main/Tasks/task6.py