IoT Device Simulator with Monitoring Dashboard 🚀

📙 Description:

In this exciting assignment, you will create a Python-based IoT device simulator that emulates the behavior of a real IoT device. The simulator should be able to handle sensor data, perform basic analytics, and communicate with a central server. To make it more engaging, you'll also build a monitoring dashboard to display the collected sensor data and analytics results. This assignment will help you sharpen your Python programming skills, including data structures, classes, OOP objects, instance and class variables, OOP methods, pip, static methods, files & standard library, exception handling, modules, and packages.

🎯 Objective:

- Develop a Python-based IoT device simulator
- Implement basic data analytics and processing
- Create an eye-catching monitoring dashboard to display sensor data and analytics results
- Use Python data structures, classes, OOP objects, instance and class variables, OOP methods, pip, static methods, files & standard library, exception handling, modules, and packages

📝 Steps:

- Create a `Sensor` class that simulates the behavior of a real-world sensor.
- Implement a `DataProcessor` class that processes and performs basic analytics on the sensor data.
- Create a `Communication` class that simulates communication with a central server.
- Build a Device class that brings together the `Sensor`, `DataProcessor`, and `Communication` classes to simulate a complete IoT device.
- Implement exception handling in the Device class for potential errors.
- Create a `Dashboard` class that displays the collected sensor data and analytics results.
- Organize your code into modules and packages to improve its maintainability and readability.
- Write tests to verify the functionality of the IoT device simulator.

🏆 Grading:

- `Sensor` class implementation (12%)
- `DataProcessor` class implementation (12%)
- `Communication` class implementation (12%)
- `Device` class implementation (12%)
- Exception handling implementation (12%)
- `Dashboard` class implementation (12%)
- Code organization using modules and packages (12%)
- Test coverage and functionality (16%)

💡 Hints:

- Use Python's `random` library to generate random sensor data.
- Use the `statistics` module for data analytics.
- Use exception handling to manage unexpected errors in the code.
- Use Python's `unittest` library to create test cases for your code.
- Use Python's `datetime` module to timestamp your sensor data.
- Implement a text-based dashboard or use a library like `matplotlib` to create a graphical dashboard.
- Organize code using packages and modules for better maintainability.
    - Create the following directory structure:

```
iot_device_simulator/
|-- __init__.py
|-- sensor.py
|-- data_processor.py
|-- communication.py
|-- device.py
|-- dashboard.py
|-- main.py
|-- tests/
    |-- __init__.py
    |-- test_iot_device_simulator.py
```

🔥 Extra:

To extend this assignment further and to get extra marks, you can consider the following enhancements:

- Integrate more types of sensors: You can create additional subclasses of the `Sensor` class that simulate different types of sensors, such as temperature, humidity, or light sensors. This will allow you to explore inheritance and polymorphism in Python.

- Add advanced data analytics: You can implement more advanced analytics methods in the `DataProcessor` class, such as moving averages, data normalization, or data filtering. This will give students experience working with Python's numerical libraries, such as NumPy or SciPy.

- Implement a graphical dashboard: You can use a Python library, such as `matplotlib` or `plotly`, to create a graphical dashboard for displaying sensor data and analytics results. This will provide students with experience in data visualization using Python.

- Simulate multiple devices: You can extend the assignment to support multiple IoT devices, each with its own sensor, data processor, and communication capabilities. This can be achieved by creating a `DeviceManager` class that manages a collection of `Device` instances. This will give students experience in working with complex systems and handling multiple instances of classes.

- Add persistence: Store sensor data and analytics results in a local file or a database, such as SQLite, using Python's built-in libraries. This will teach you how to work with file I/O and databases in Python.

Good luck with this assignment, and have fun exploring the world of IoT devices using Python! 💯💥