Logging Module in Python: Comprehensive Guide with Real-time Example

Key Features of the `logging` Module

1. Log Levels:

   - DEBUG: Detailed information, primarily for debugging.

   - INFO: General information about program execution.

   - WARNING: Something unexpected but not critical.

   - ERROR: Serious problems that prevent parts of the program from working.

   - CRITICAL: Very serious issues indicating the program may not continue running.

2. Log Components:

   - Loggers: Entry points for your code to send log messages.

   - Handlers: Decide where the log messages go (e.g., console, file, etc.).

   - Formatters: Specify the layout of log messages.

3. Flexible Configuration: Supports logging to multiple destinations (console, files, external services).

Basic Logging Configuration

```
import logging

logging.basicConfig(
    filename='application.log',
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)


logging.debug("Debug message")

logging.info("Info message")

logging.warning("Warning message")

logging.error("Error message")

logging.critical("Critical message")
```

Output in `application.log`:

```
2024-12-18 10:00:00 - root - DEBUG - Debug message

2024-12-18 10:00:01 - root - INFO - Info message

2024-12-18 10:00:02 - root - WARNING - Warning message

2024-12-18 10:00:03 - root - ERROR - Error message

2024-12-18 10:00:04 - root - CRITICAL - Critical message
```

Real-time Case Study: MariaDB Connection and Data Manipulation

This example demonstrates how to:

1. Connect to a MariaDB database.

2. Fetch data using SQL queries.

3. Perform data manipulations.

4. Use logging to monitor and debug the workflow.

```
import logging

import mariadb

import pandas as pd
```

```
# Configure logging

logging.basicConfig(

    filename='mariadb_operations.log',

    level=logging.DEBUG,

    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'

)

logger = logging.getLogger("MariaDBLogger")



def connect_to_mariadb():

    try:

        logger.info("Attempting to connect to MariaDB...")

        conn = mariadb.connect(

            user="your_username",

            password="your_password",

            host="localhost",

            port=3306,

            database="your_database"

        )

        logger.info("Successfully connected to MariaDB.")

        return conn

    except mariadb.Error as e:

        logger.error(f"Error connecting to MariaDB: {e}")

        raise



def fetch_data(conn):

    try:
        query = "SELECT * FROM your_table;"

        logger.debug(f"Executing query: {query}")

        df = pd.read_sql(query, conn)

        logger.info("Data fetched successfully.")

        logger.debug(f"Fetched data: {df.head()}")

        return df

    except Exception as e:

        logger.error(f"Error fetching data: {e}")

        raise



def manipulate_data(df):

    try:

        logger.info("Starting data manipulation...")

        df['new_column'] = df['existing_column'] * 2

        logger.debug(f"Data after manipulation: {df.head()}")

        logger.info("Data manipulation completed successfully.")

        return df

    except KeyError as e:

        logger.error(f"Column missing: {e}")

        raise

    except Exception as e:

        logger.error(f"Error during manipulation: {e}")

        raise



def main():

    conn = None

    try:
        conn = connect_to_mariadb()

        data = fetch_data(conn)

        manipulated_data = manipulate_data(data)

        logger.info("Process completed successfully.")

    except Exception as e:

        logger.critical(f"Unhandled exception: {e}")

    finally:

        if conn:

            conn.close()

            logger.info("Database connection closed.")


if __name__ == "__main__":

    main()
```