Date

Problem #02

The logging DISASTER:-

In real systems, many things happen at the same time, and when something goes wrong, you need to reconstruct what happened, in what order and where.

Imagine 8 STEP Pipeline working In Production. And STEP 4 fails. Only 1 out of 10,000 batches, at 2:13 Pm, only in production when 2 workers run in parallel.

How Do you know?

⇒ That is logging problem.

logging is about recording events, not printing "messages".

Learning:

How to design and configure a single logging system that the entire application can rely on.

Not to write a logger from stratch, but to use python's logging correctly.

Date

Strategy:

Build a wrapper, or configuration Model.
Because python already gives us Thread

→ safe logging.
→ log ~~Event~~ levels
→ Formatters
→ Handlers
→ File rotation
→ console output

Reinventing that would be a mistake

Requirements

↳ logging Levels
  ↳ debug
  ↳ info
  ↳ warning
  ↳ Error
  ↳ Critical

basicConfigic ( )

↳ logging.basicconfig (level = logging.DEBUG)

⇒ you can also set this logging level from a Database.

Date

Other Arguments of basicConfig()

↳ You can set the fileName, If you want to write to a log to a file

↳ You can change the way logging is formatted.

Logging Handler:

→ Handler decides where the log goes.

→ streamHandler takes log to console

→ FileHandler takes log to file

→ RotatingFileHandler

→ TimedRotatingFileHandler

→ SMTPHandler sends logs via Email

→ HTTPHandler sends logs to remote via HTTP

→ socketHandler sends logs over sockets

→ WatchedFileHandler watches file changes.

We either use basicConfig or Handlers

basic.config() = quick and dirty setup

Handlers = real engineering setup.

Date

basicConfig() vs Handlers

basicconfig() Implicitly creates and configures a single handler for the root logger.
If no filename is provided, it defaults to a console (stream) Handler and print logs on console. If we do provide a filename. It sets the Handler to a FileHandler and writelogs to a file.

Because it only supports one implicit handler and basic configuration. It is only suitable only simple scripts.

For scalable and complex systems, we explicitly create and manage handlers, which gives us full control, extensibility and correctness.

For Real systems, DO NOT Rely on Basicconfig at All. DO everything through Handlers.

→ DO not pass loggers around Everywhere.
→ DO configure logging once.
→ DO get loggers locally via getLogger(__name__)

Date

Logging has Two Phase

1- Configuration Phase

"How logging behaves Globally"

This includes

- ↳ Handlers (console, file, etc).
- ↳ Formatters
- ↳ log levels
- ↳ Propagation Rules.

All of this happens once at application startup.

2. Usage Phase

"Who is emitting Module"

This includes

- ↳ logger.info(...)
- ↳ logger.error(...)

This happen in every Module.

CODING PART

Python logging has 4 core concepts

1- logger
2- Handler
3- Formatter
4- Output Level

Logger → Handler → Formatter → output

Date

Python's logging Module

we mainly use

- ↳ logging.getLogger(name)
- ↳ logging.Logger
- ↳ logging.Handler
- ↳ logging.FileHandler
- ↳ logging.StreamHandler
- ↳ logging.Formatter.

Logger. Methords

- ↳ logger.debug()
- ↳ logger.info()
- ↳ logger.warning()
- ↳ logger.error()
- ↳ logger.critical()

STEP 02: Correct Architecture

logging_config.py ← configures logging once
main.py ← uses logger
Ingestion.py ← uses logger.

STEP 03: logging configuration (Handlers way)

logging_config.py

```
Import logging
def setup_logging():
```

#1. create formatter (How logs look)

```
formatter = logging.formatter( "%(asctime)s |
        %(levelname)s | %(message)s " )
```

#2. Console handler (stdout)

```
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.INFO)
console_handler.setFormatter(formatter)
```

#3 File Handler

```
File_handler = logging.FileHandler("APP.log")
File_handler.setLevel( logging.DEBUG)
File Handler.setformatter (formatter)
```

#4. root logger (Global)

```
root_logger = logging.getLogger()
root_logger.setLevel(logging.DEBUG)
```

#5 ATTACH Handlers (critical)

```
root_logger.addHandler (console_handler)
rootlogger.addHandler (File_handler)
```

STEP 04: Using Logging

main.py

```
Import logging
from logging_config import setup_logging()
```

Date

```
# Configure logging once
setup_logging()

# Get module-specific logger
logger = logging.getLogger(__name__)

logger.info("Application STARTED")
logger.debug("Debugging Details Here")
logger.warning("This is warning")
logger.error("This is error")
logger.critical("This is critical").
```

Key Rules

1. ↳ logging is configured once, used Everywhere.
2. ↳ Never call basicConfig and Handlers together
   Pick one approach
3. ↳ Never Pass loggers as perimeters
4. ↳ Every file uses
   logger = logging.getLogger(__name__)

Key Points

- Every logger you create with getlogger(name) is a child of root logger unless you explicitly break propagation.
- If a Child logger (logger = logging.getlogger("Module1") has (propagation = True) Default, then any log it emits goes through its own handlers first, (if any) and then bubbles up to its root handler
- __name__ is a python special variable

```
# main.py
Print(__name__)        # main

# helpers.py
print(__name__)        # helpers.
```