📅 DAY 04 — Context Managers (`with`)

Goal : Safely manage resources

(files, connections, locks, data streams)

1️⃣ The Problem (Why Context Managers Exist)

❌ Bad way (manual cleanup)

```
file = open("data.txt","r")
data = file.read()
file.close() # easy to forget
```

if an error happens → file may never close ❌

2️⃣ The Pythonic Solution (`with`)

✅ Good way

```
with open("data.text","r") as file:
    data = file.read()
```

✔️ File is automatically closed

✔️ Even if an error happens

✔️ Cleaner and safer

Rule:

If you see `with`, Python is handling cleanup for you.

3️⃣ What Is a Context Manager (Simple Definition)

A **context manager** is something that:

- **Enters** a block (`__enter__`)
- **Exits** a block (`__exit__`)

- Cleans up resources automatically

`with` = "enter → do work → exit safely"

4️⃣ Real-Life Analogy 🚪

Think of a context manager like:

- Opening a door
- Doing work inside a room
- Closing the door when you leave

Python guarantees the door closes.

5️⃣ Using `with` for Common Tasks

Reading a file:

```
with open("data.txt") as f:
    for line in f:
        print(line)
```

Writing to a file:

```
with open("log.txt", "w") as f:
    f.write("training started")
```

Multiple context Mangers

```
with open("a.txt") as f1, open("b.txt") as f2:
    print(f1.read(), f2.read())
```

6️⃣ Creating Your Own Context Manager (Simple Way)

Step 1: Using a class

```
class MyContext:
    def __enter__(self):
        print("Entering context")
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        print("Exiting content")
```

Usage:

```
with MyContext():
    print("Doing work")

# output
Entering context
Doing work
Exiting content
```

7️⃣ Easier Way (Recommended): `contextlib`

Python gives a shortcut

```
from contextlib import contextmanager

@contextmanager
def my_context():
    print("start")
    yield
    print("End")
```

Usage:

```
with my_context():
    print("Working")

# output
start
Working
End
```

8️⃣ ML Engineer Example

Reading large datasets safely:

```
def data_loader(file_path):
    with open(file_path) as f:
        for line in f:
            yield line.strip()
```

This combines:

✔️ context managers

✔️ generators

✔️ memory efficiency