📆 DAY 06 — Modules & Packages

Goal : Organize code like a real project

Think of today as:

> “How Python finds and runs your code across files”

1️⃣ What Is a Module?

A Module is a python file.

Example:

```
math_utils.py
```

```
# math_utils.py
def add(a, b):
    return a + b
```

Use it:

```
import math_utils

print(math_utils.add(2,3))
```

✔️ One file = one module

2️⃣ Different Ways to Import

Import specific module

```
import math_utils
math_utils.add(1,2)
```

Import specific function

```
from math_utils import add
add(1,2)
```

Import with alias(very common)

```
import numpy as np
```

3️⃣ What Is a Package?

A package is a folder with Python files.

Example:

```
utils/
 ├── __init__.py
 ├── math_utils.py
 ├── file_utils.py
```

Now you can do :

```
from utils.math_utils import add
```

📌 `__init__.py` tells Python:

> "This folder is a package"

4️⃣ Why Packages Matter (ML Context)

ML projects have many parts:

- Data Loading
- Preprocessing
- Training
- Evaluation

Packages keep this clean.

Example:

```
ml_project/
 ├── data/
 ├── models/
 ├── training/
 ├── utils/
```

5️⃣ The `__main__` Concept (Very Important)

```
# my_script.py
def main():
    print("Running main")

if __name__=="__main__":
    main()
```

Why this matters

- Code runs only **when file is executed directly**
- Prevent code from running during imports

Used everywhere in real projects.

6️⃣ Python Path (Simple Explanation)

Python searches imports in:

1. Current folder
2. Installed packages
3. `PYTHONPATH`

👉 That's why project structure matters.

7️⃣ Common Beginner Mistakes ❌

❌ Circular imports

❌ Messy file names

❌ Running logic at import time

❌ No `__main__` guard

We avoid all of these.