**1. Python Modules**

**What is a Module?**

* A **module** in Python is simply a **Python file** (.py) that contains **functions, variables, and classes** you can use in other Python files.
* Example:  
  If you have a file called math\_utils.py:

def add(a, b):

return a + b

def subtract(a, b):

return a - b

You can reuse its code in another file using import.

**Why Use Modules?**

✅ Reusability → Write once, use multiple times.  
✅ Organization → Break large programs into smaller, manageable files.  
✅ Maintainability → Easier to fix or update specific parts of a program.  
✅ Collaboration → Multiple people can work on different modules.

**2. Python Imports**

**How to Import a Module**

* **Basic import**

import math\_utils

print(math\_utils.add(5, 3))

* **Import with alias**

import math\_utils as mu

print(mu.add(5, 3))

* **Import specific functions**

from math\_utils import add

print(add(5, 3))

* **Import all functions (not recommended)**

from math\_utils import \*

print(add(5, 3))

**Built-in Modules**

Python has many built-in modules like:

import math

import random

import datetime

You can check all built-in modules with:

help("modules")

**3. \_\_init\_\_.py File**

**What is \_\_init\_\_.py?**

* A special file that tells Python **“This directory is a package.”**
* Without it (in Python < 3.3), Python wouldn’t treat the folder as a package.
* In modern Python (3.3+), it’s **optional**, but still **used for organization and initialization code**.

**Example Package Structure**

my\_package/

\_\_init\_\_.py

math\_utils.py

string\_utils.py

**Using the Package**

from my\_package import math\_utils

print(math\_utils.add(5, 3))

**What can \_\_init\_\_.py do?**

* Initialize variables or data when the package is imported.
* Control what’s accessible when you do:

from my\_package import \*

* Example \_\_init\_\_.py:

from .math\_utils import add

from .string\_utils import capitalize\_words

\_\_all\_\_ = ["add", "capitalize\_words"] # Controls wildcard import

**4. Types of Imports**

**Absolute Import**

Full path from the project’s root:

from my\_package.math\_utils import add

**Relative Import**

Import relative to the current file:

from .math\_utils import add # same folder

from ..utils import helper # one folder up

**5. Advantages of Using Modules & Imports**

* **Code Reuse** → Avoid rewriting the same logic.
* **Cleaner Structure** → Each file has a specific purpose.
* **Easier Maintenance** → Fix bugs in one place, update everywhere.
* **Namespace Management** → Avoid variable/function name conflicts.
* **Scalability** → Suitable for large projects.

**6. Import Tips & Best Practices**

* Use **specific imports** instead of \* (wildcard) to avoid name clashes.
* Keep **module names lowercase** for readability.
* Group imports:

# Standard library imports

import os

import sys

# Third-party imports

import requests

# Local application imports

from my\_package import math\_utils

* Avoid **circular imports** (when two modules import each other).

**Quick Recap Table**

| **Concept** | **Meaning** | **Example** |
| --- | --- | --- |
| **Module** | .py file containing code | math\_utils.py |
| **Package** | Folder with \_\_init\_\_.py and modules | my\_package/ |
| **Import** | Load and use module/package | import math\_utils |
| **\_\_init\_\_.py** | Marks a folder as a package & runs init code | my\_package/\_\_init\_\_.py |
| **Absolute Import** | Full path import | from my\_package import file |
| **Relative Import** | Relative path import | from .file import func |