📘 1. Step-by-Step Guide: Installing Your Local Python Module (Option 2)
Goal
Install your own Python module into your projects without publishing it to PyPI, using .

Step 1 — Create your module structure
Example:

Your inner folder () is the actual package.

Step 2 — Create a file
Inside the top-level folder:

Step 3 — Install your module locally
Navigate to the folder containing :

Install in editable mode:

Editable mode means:
- You can edit the module's code
- All projects using it see updates instantly
- No reinstall needed

Step 4 — Import it in any project
Now you can use it anywhere:

Step 5 — Update your module anytime
Just edit the files inside your module folder.
No reinstall needed because of .

Step 6 — Optional: Version control
Put your module under Git:

This keeps your reusable code clean and trackable.

📚 2. Full Documentation-Style Version
Below is a polished, professional documentation page suitable for a PDF or internal developer handbook.

Using a Local Python Module Across Multiple Projects (Without Publishing to PyPI)
Overview

This document describes how to install and reuse a private Python module across multiple projects using a local installation method. This approach avoids publishing to PyPI while still providing a clean, maintainable workflow.

1. Directory Structure
A typical module layout:

Key points
- The outer folder is the project root.
- The inner folder is the actual importable package.
- defines how the module is installed.

2. Creating the File
A minimal but complete :

Fields
- name: The import name used by Python.
- version: Helps track updates.
- packages: Automatically includes all subpackages.

3. Installing the Module Locally
Navigate to the module's root directory:

Install using pip:

Why (editable mode)?
- The module is linked, not copied.
- Any code changes are immediately reflected in all projects.
- Ideal for active development and internal libraries.

4. Using the Module in Your Projects
Once installed, import it normally:

Your module behaves exactly like a PyPI package — without being published.

5. Updating the Module
Because editable mode is used:
- Modify the source files directly
- No reinstall required
- All dependent projects automatically see updates

This makes the workflow extremely efficient for internal tools and shared utilities.

6. Optional: Version Control
To maintain history and collaborate:

You may later push this to a private Git server if desired.

7. Summary

Using provides:

- A clean installation method
- No need for PyPI
- Automatic updates across projects
- Professional packaging structure
- Compatibility with all Python tooling

This is the recommended approach for internal modules, automation libraries, and reusable codebases.

If you want, I can also prepare:

- A PDF-ready cover page
- A table of contents
- A multi-module architecture guide
- A best practices section for internal Python libraries

Just tell me how polished you want your documentation to be.