

Why packaging matters:

Packaging matters because it solves the “it works on my machine” problem. Without setup.py, every developer has to manually install the right version of Flask, psycopg, beautifulsoup4, etc, and hope they match.

Setup.py declares the project as an installable Python package. This gives

- Importability – editable install means import app, import scrape, etc work from anywhere, not just when you cd into src/. No more sys.path hacks. Without packaging, Python can’t find modules unless in the right directory or hack sys.path. With it, import app works everywhere
- Dependency management – install_requires and extras_require let pip resolve and install everything in one command instead of manually tracking requirements.txt. pip handles version conflicts automatically
- Reproducibility – anyone can clone the repo, run pip install -e “[dev]”, and get identical environment with all runtime + dev dependencies.
- Distribution – if you ever need to pip install the projects in a Docker container or CI, it just works. CI already uses pip install -e “[dev]”.
- Single command setup – pip install -e “[dev]” installs everything. No guessing.
- CI reliability – Github Actions workflow uses pip install -e “[dev]”. Without setup.py, CI wouldn’t know what to install.
- Separation of concerns – install_requires lists what the app needs to run. Extras_require[“dev”] adds what developers need (pytest, pylint). Production deployments skip dev tools.

In short, it turns a collection of scripts into a proper python package with declared dependencies, making installation, testing and deployment reliable.

The alternative – a bare requirements.txt with pip install -r requirements.txt. This doesn’t make the code importable as a package, doesn’t support extras and doesn’t declare metadata. It’s fine for scripts but not for multi-module projects.