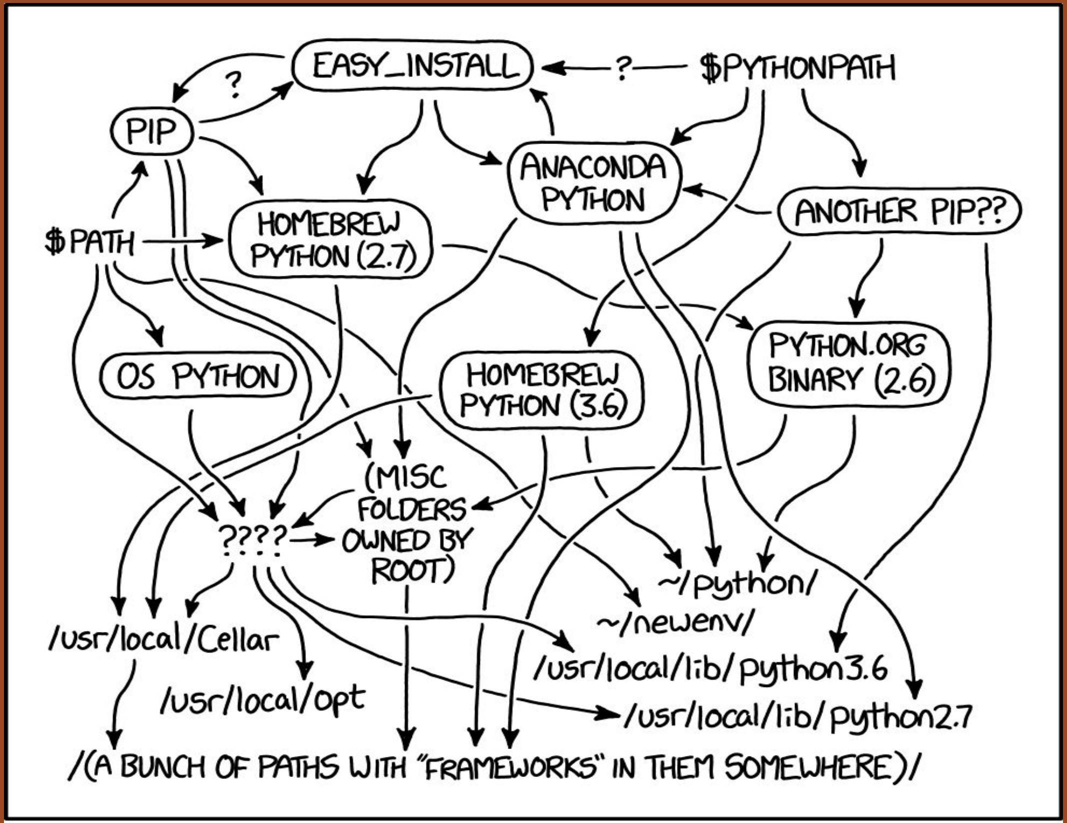
Python Packaging with Pixi

Priyanka Ojha
https://orcid.org/0000-0002-6844-6493

v1.0
21/10/2025

Agenda

1. Why
2. What
3. How (Outline + Demo + Hands On)
4. Summary
5. Q&A



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Ref : https://xkcd.com/1987/

Why Python Packages?

1. To share your code in an easy to install way
2. To make it reusable
3. To manage dependencies
4. To collaborate and allow others to build on it.
5. To publish it through the official Python Package Index (PyPI) for trusted distribution and discovery.
6. To make your code more reliable and maintainable through versioning, metadata, and packaging best practices.
7. Packaging is all about target environment and deployment experience. Ref : https://packaging.python.org/en/latest/overview/#thinking-about-deployment

What is a Python Package

A Python package is a way to package and distribute your code to the world. Its best to consider it as a beautiful wrapper around your code which you present to the world usually via a PyPI , conda-forge, Bioconda etc.

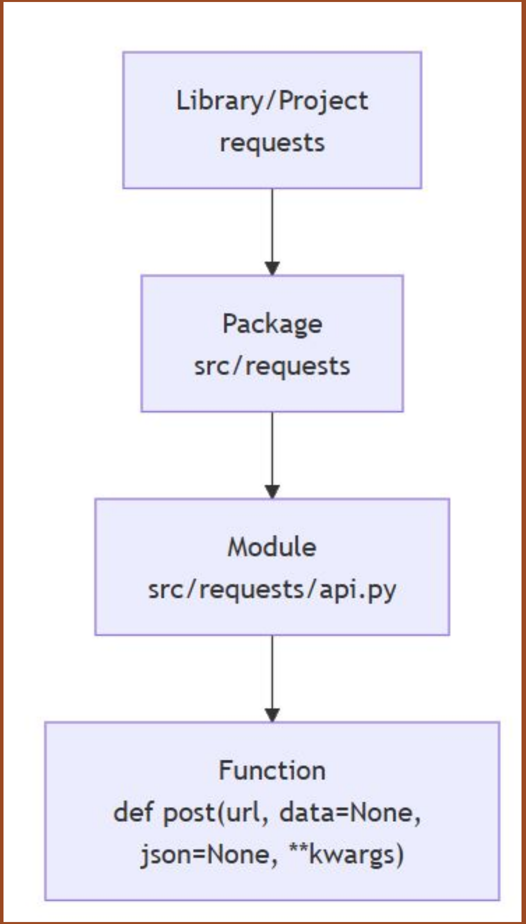
function vs module vs package vs library vs Python package

1. **Function :** A block of reusable code that performs a single specific task.
2. **Module :** A single Python file (.py) that groups related classes, functions and variables.
3. **Package :** A folder/ collection containing multiple modules and an __init__.py file to organize them.
4. The __init__.py files are required to make Python treat folders containing the file as packages
5. **Library :** A collection of related packages or modules that provide broad functionality for reuse.
6. **Library** can have **Package(s),** which can have **module(s)** which can have **function(s).** It can also be considered a Project.
7. **Python package :** A Python package is a collection of related code modules (files) bundled with metadata describing how the package should be installed and used *

* **Ref :** https://pydevtools.com/handbook/explanation/what-is-a-python-package/

Example : requests

1. https://pypi.org/project/requests/
2. https://github.com/psf/requests



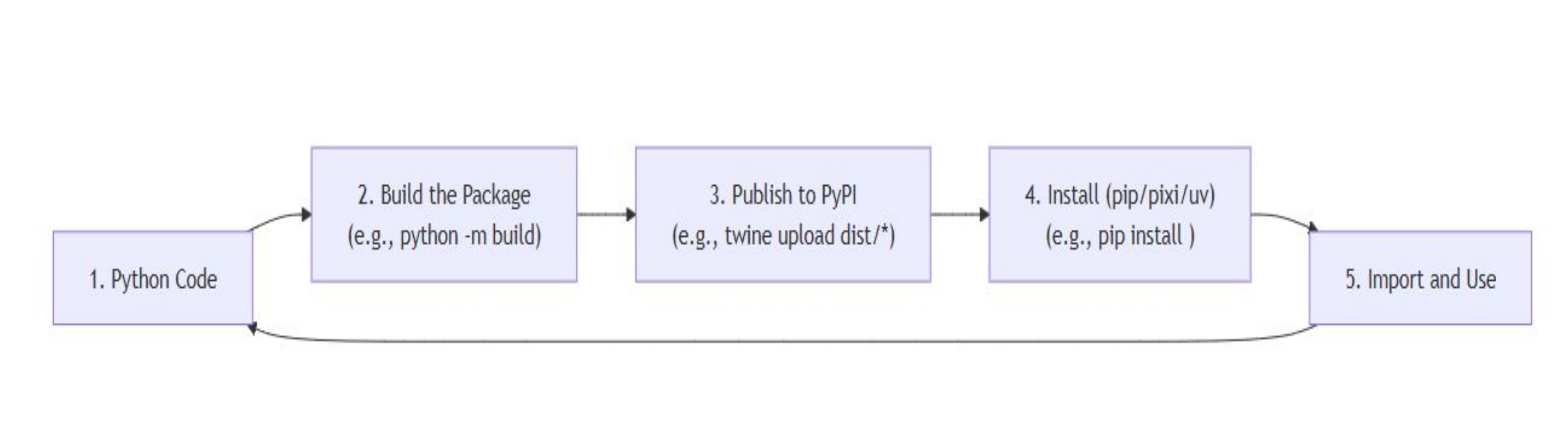
Library/Project
requests

Package
src/requests

Module
src/requests/api.py

Function
def post(url, data=None, json=None, **kwargs)

Steps to create a Python Package


1. Python Code
2. Build the Package (e.g., python -m build)
3. Publish to PyPI (e.g., twine upload dist/*)
4. Install (pip/pixi/uv) (e.g., pip install)
5. Import and Use

Many ways/tools to create Python Packages

1. setup.py
2. setup.cfg -> pyproject.toml
3. poetry
4. uv
5. pixi

Why pixi ?

1. Support for both **PyPI** and **Conda** packages : enabling flexibility in sourcing dependencies.
2. **Performance** :lightweight and modern, designed for speed.
3. **Multi-language dependency management** : e.g. Python with Rust, or Python with C/C++
4. **Integration with uv** : leveraging a high-performance package installer
5. **Reproducibility**: guaranteed through the use of **pixi.lock**
6. Configuration via TOML files: **supports** both **pixi.toml** and **pyproject.toml**
7. Conda packages are very popular in scientific communities.

For our demo we will be using **pyproject.toml**

Ref : https://pixi.sh/dev/#why-pixi

Why pyproject.toml ?

1. pixi.toml is not compliant with PEP 621, PEP 517, or PEP 660.
2. Pixi (the package manager that uses pixi.toml) is designed to integrate with and respect these Python standards, by supporting a pyproject.toml file when managing Python projects.
3. The pyproject.toml file is a standard for Python projects.

Ref : https://pixi.sh/v0.40.1/reference/pixi_manifest/#pypi-dependencies

https://pixi.sh/v0.40.1/advanced/pyproject_toml/

Demo using pixi

Follow the Tutorial from via this link or QR Code or bitly link.

https://priya-gittest.github.io/Python-Packaging-with-Pixi/

http://bit.ly/42Q5eEX

Summary

- Packaging your Python code makes it easy to share ,reuse and installable across projects.
- Follow the sequence : **Code → Build → Publish→ Install → Import**
- Always follow the latest guidelines and best practices from the **Python Packaging Authority (PyPA)** to ensure your packages are compatible, maintainable, and installable via PyPI.
- Follow latest pixi documentations for any code changes /improvements.

Credits & License

Author: Priyanka Ojha, https://orcid.org/0000-0002-6844-6493