Python Packaging

A Brief Recap of the PYOPP Workshop

Anno Knierim

aknierim

July 1, 2025

Why Even Bother With Packaging?

Packages allow you to share your code, so other people can use it.

But also...

- Helps you keeping your code from breaking
- Benefits other people that may have faced a similar problem
- Saves time because code can be reused easily

How Do I Create a Package?

There does not exist “the one way” to create packages, but...

- Modern packaging uses a scaffolding called `pyproject.toml` with three important sections:

`[build-system]`	Allows you to describe what build backend to use.
`[project]`	Sets up metadata for the package, such as the name or version.
`[tool]`	A section for tool configuration.

- An easy way to set up that scaffolding: `hatch`

```
$ pip install hatch
$ mamba install hatch
```

How Do I Create a Package?

- Use hatch's CLI tool to quickstart creating a package:

```
$ hatch new my_package
```

- Let's see what this created:

```
$ head my_package/pyproject.toml
```

- You can also upgrade an existing project to use `hatch`:

```
$ hatch new --init
```

- Have a look at the Writing your `pyproject.toml` guide to learn how to customise the `pyproject.toml` file

Output

```
my_package
├── src
│   └── my_package
│       ├── __about__.py
│       └── __init__.py
├── tests
│   └── __init__.py
├── LICENSE.txt
├── README.md
└── pyproject.toml
```

```toml
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "my_package"
dynamic = ["version"]
description = ''
readme = "README.md"
requires-python = "⩾3.8"
```

Dependencies

- Dependencies for your project are defined with the `dependencies` key inside the `[project]` section
- You can set dependency specifiers (aka constraints) such as versions
- Define your optional dependencies in the `[project.optional-dependencies]` section and group them
- Install optional dependencies using

  ```
  $ pip install my_package[plot]
  ```

- Fairly new (accepted `2024-10-10`): PEP 735 Dependency Groups
- Optional dependencies that are ***not*** installed when a ***user*** installs the package, e. g., via PyPI
- Install the groups from within your source repo:

  ```
  $ pip install --group dev
  ```

Example

```toml
[project]
dependencies = [
  "numpy",
  "astropy ⩽ 6.1.0",
  "tomli;python_version<'3.11'",
]
```

```toml
[project.optional-dependencies]
plot = ["matplotlib"]
```

```toml
[dependency-groups]
tests = ["pytest", "pytest-cov"]
docs = ["sphinx"]
dev = [
  "jupyter",
  "pre-commit",
  {include-group = "tests"},
  {include-group = "docs"},
]
```

CLI Scripts

- We can expose scripts in our package using the `pyproject.toml` `[project.scripts]` section
- Similarly: Entry points, that allow the creation of plugins, and cross-platform compatibility
  - See Entry Points

Example

src/my_package/cli.py:

```python
def print_message():
    print("Hello World!")
    raise SystemExit(1)
```

pyproject.toml:

```toml
[project.scripts]
hello-world =
    "my_package.cli:print_message"
```

Result

```
$ hello-world
Hello World!
```