UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Python Packaging

Liam Keegan, SSC

Motivation

It would be reasonable to assume that packaging Python code should be a straightforward, well-defined, user-friendly process, much like writing Python code...

...unfortunately this is not the case, and some googling will provide a multitude of conflicting advice on how to do this.

In fact the Python Packaging User Guide (*packaging.python.org*) begins with this slightly discouraging note:

> *"Building your understanding of Python packaging is a journey. Patience and continuous improvement are key to success."*

Our goal today is to present a modern approach to packaging and distributing your Python code.

Course Outline

- Python package
  - Source file layout
  - Metadata in pyproject.toml
- PyPI
  - Publish a package
  - Automated deployment
- Conda-forge
  - Write a recipe
- Compiled extension modules
  - Use pybind11 / nanobind c++ libraries
  - Use scikit-build-core for CMake integration
  - Use cibuildwheel to automate building of binary wheels

A Minimal Python Package

What is a Python module?

A Python module is something that we can import, typically a single Python source file (excluding the .py filename extension)

- Assume a Python source file named stats.py which contains a function
  - `mean(x)`
- We can import the module in Python:
  - `import stats`
- And then use the function defined in this module:
  - `stats.mean([1, 2, 3])`

If you have many modules it is often convenient to organise them in a hierarchical way, which can be done using a Python package.

What is a Python package?

A Python package is simply a folder that contains Python source code files, including one with the special name “__init__.py”.

When you import a package, the contents of __init__.py are automatically imported (this file can be and often is empty).

- calculate/
  - __init__.py
  - stats.py
- We can import this package in Python:
  - `import calculate`
  - this imports any code in __init__.py
- And then use the stats module from our calculate package:
  - `calculate.stats.mean([1, 2, 3])`

Note that packages can contain sub-packages, and a package is also a module.

Metadata

In addition to our Python code, to make a package usable by others we typically need to provide a lot of additional information, such as:

- Which Python versions are supported
- Which other Python libraries need to be installed
- Any non-Python dependencies (e.g. system libraries, compiled code)
- Test code and how to run it
- Documentation
- License
- Entry points

This information is provided by the pyproject.toml file

pyproject.toml

Todo...

Recommended project structure: src layout

```
/calculate
    /src
        /calculate
            __init__.py
            add.py
            ...
    pyproject.toml
```

Any code that should be importable goes here - this is the Python package

Defines how to install the package

See https://packaging.python.org/en/latest/discussions/src-layout-vs-flat-layout for more information

Recommended project structure: src layout

```
/calculate
    /src
        /calculate
            __init__.py
            add.py
            ...
    /tests
        __init__.py
        test_add.py
        ...
    /docs
        ...
    pyproject.toml
    README.md
    LICENSE
```

Any code that should be importable goes here - this is the Python package

The test suite and documentation are part of the project, but not part of the Python package

Defines how to install the package

All projects should have a readme

All projects should have a license

See https://packaging.python.org/en/latest/discussions/src-layout-vs-flat-layout for more information

Installation

User clones your git repository, or downloads and extracts a zip file, or in some way is provided with these files. Then they install your package with the command:

- `pip install .`

Python Package Summary

- /src folder
  - Python package that is to be installed
- /tests folder
  - Automated test suite for the package
- /docs folder
  - Documentation for the package
- pyproject.toml
  - Metadata and dependencies required for installation
- README
- LICENSE

Given these files, a user can simply install your package with `pip install .`

PyPI: Python Package Index

What is PyPI?

A central index of Python packages.

Instead of the user downloading your package and then installing it, you can upload your package to PyPI and the user can directly install it from there:

```
pip install calculate
```

Metadata in pyproject.toml

Todo: what is required, optional for PyPI

Wheel / twine

How to manually build & publish a package

GitHub actions

Automatic publishing using CI

Conda-forge

Conda-forge

Conda is a package manager, and not just for Python packages.

Anaconda, conda-forge, bioconda, etc are repositories of conda packages known as channels.

Conda-forge dependencies

For PyPI, list all python dependencies, but have to bundle any non-python compiled dependencies in binary wheels.

On conda-forge can also have binary/compiled dependencies, however *all* dependencies must be on conda-forge, cannot install them from pypi!

recipe

Todo

Pull request

Todo

C++ bindings

C++ bindings

Common use case is to provide a way to use compiled c++ code from Python.

Typically this is for performance reasons, but can also be done to provide a Python interface to existing legacy c++ code without having to rewrite it in Python.

Complication is that the code needs to be compiled separately for each version of Python and for each combination of Operating system and architecture.

pybind11

Defacto standard for providing python bindings from c++ code

nanobind

Faster re-write of pybind11, less mature

CMake

scikit-build-core

Scientific Software Center

cibuildwheel

Automatic compilation of wheels for all python versions, operating systems and architectures

GitHub actions

How to integrate all this

conda-forge

Scientific Software Center

Summary

Summary

In this course we covered:

- Testing
  - What is automated testing
  - Benefits of a good test suite
  - Different types of tests
- Hands on with pytest
  - Install and run pytest
  - Write simple tests
  - Use temporary files in tests
  - Use fixtures to manage resources
  - Parametrize tests
  - Write tests in a Jupyter notebook
- Best practices
  - How to write good tests

Next steps

- packaging.python.org
  - Great resource on Python packaging
- Simple example of Python package with c++ extension
- For your next Python package
  - Our cookiecutter generates a Python package with CI for wheel building and PyPI publishing:
    - github.com/ssciwr/cookiecutter-python-package