Custom class name

John Strickler

Version 1.0, July 2023

Table of Contents

Chapter 1: Packaging

Objectives

- Learn about packages vs apps
- Write setup files
- Understand the types of wheels
- Know when to create a non-wheel distribution
- Create a reusable package
- Configure dependencies
- Configure executable scripts
- Distribute and deploy packages

NOTE This chapter is a work in progress. There may be missing parts, or errors in code or links.

How to package (the old way)

- Start with standard layout
- Add configuration files
- Use setuptools or build tools to build package

To create a Python package, start with a good standard project layout. Add configuration and other files to the top level of the project.

Create the `setup.py` configuration module (which calls the **setup** function.

Use **setuptools** or **build** to build the package as an installable source or binary package.

Package files with `setup.py`

- README.rst reStructuredText setup doc
- MANIFEST.in list of all files in package
- LICENSE s/w license
- setup.py controls packaging and installation

There are four files to create in the package.

README.rst is a typical README file, that describes what the package does in brief. It is not the documentation for the package. It should tell how to install the package. Is is usual to use reStructuredText for this file, hence the .rst extension.

MANIFEST.in is a list of all the non-Python files in the package, such as templates, CSS files, and images. It should also include LICENSE and README.rst.

LICENSE is a file describing the license under which you're releasing the software. This is less important for in-house apps, but essential for apps that are distributed to the public.

The most import file is **setup.py**. This is a Python script that uses setuptools to control how your application is packaged for distribution, and how it is installed by users.

setup.py contains a call to the setup() function; named options configure the details of your package.

TIP The following link has some great suggestions for laying out the files in a package: https://blog.ionelmc.ro/2014/05/25/python-packaging/

Overview of setuptools

- Creates distributable files
- Can be used for modules or packages
- Many distribution formats
- Configuration in setup.py

The key to using setuptools is setup.py, the configuration file. This tells setuptools what files should go in the module, the version of the application or package, and any other configuration data.

The steps for using setuptools are:

- write a setup script (setup.py by convention)
- (optional) write a setup configuration file
- create a source, wheel, or specialized built distributions

The entire process is described at https://packaging.python.org/distributing/.

See https://packaging.python.org/glossary/#term-built-distribution for a glossary of packaging and distribution terms.

Preparing for distribution

- Organize files and folders
- Create setup.py

The first step is to create setup.py in the package folder.

setup.py is a python script that calls the setup() function from setuptools with keyword (named) arguments that describe your module.

For modules, use the py_modules keyword; for packages, use the packages.

There are many other options for fine-tuning the distribution.

Your distribution should also have a file named README or README.txt which can be a brief description of the distribution and how to install its module(s).

You can include any other files desired. Many developers include a LICENSE.txt which stipulates how the distribution is licensed.

Table 1. Keyword arguments for setup() function

Keyword	Description
`author`	Package author's name
`author_email`	Email address of the package author
`classifiers`	A list of classifiers to describe level (alpha, beta, release) or supported versions of Python
`data_files`	Non-Python files needed by distribution from outside the package
`description`	Short, summary description of the package
`download_url`	Location where the package may be downloaded
`entry_points`	Plugins or scripts provided in the package. Use key console_scripts to provide standalone scripts.
`ext_modules`	Extension modules needing special handling
`ext_package`	Package containing extension modules
`install_requires`	Dependencies. Specify modules your package depends on.
`keywords`	Keywords that describe the project
`license`	license for the package
`long_description`	Longer description of the package
`maintainer`	Package maintainer's name
`maintainer_email`	Email address of the package maintainer
`name`	Name of the package
`package_data`	Additional non-Python files needed from within the package
`package_dir`	Dictionary mapping packages to folders
`packages`	List of packages in distribution
`platforms`	A list of platforms
`py_modules`	List of individual modules in distribution
`scripts`	Configuration for standalone scripts provided in the package (but entry_points is preferred)
`url`	Home page for the package
`version`	Version of this release

Creating a source distribution

- Use setup.py with -sdist option
- Creates a platform-neutral distribution
- Distribution has its own setup.py

Run setup.py with your version of python, specifying the -sdist option. This will create a platform-independent source distribution. python setup.py sdist

```
ls -l dist
total 4
-rw-rw-r-- 1 jstrick jstrick 633 2012-01-11 07:49 temperature-1.2.tar.gz

tar tzvf dist/temperature-1.2.tar.gz
drwxrwxr-x jstrick/jstrick   0 2012-01-11 00:26 temperature-1.2/
-rw-rw-r-- jstrick/jstrick 256 2012-01-11 00:26 temperature-1.2/PKG-INFO
-rw-rw-r-- jstrick/jstrick 285 2012-01-10 13:36 temperature-1.2/setup.py
-rw-r--r-- jstrick/jstrick 342 2012-01-10 07:52 temperature-1.2/temperature.py
```

To install a source distribution, extract it into any directory and cd into the root (top) of the extracted file structure. Execute the following command:

```
python setup.py install
```

Example

temperature/setup.py

```
from setuptools import setup

# note -- this file is not used when building with pyproject.toml

if __name__ == "__main__":
    setup()
```

```
cd temperature
python setup.py sdist
running sdist
running egg_info
writing temperature.egg-info/PKG-INFO
writing top-level names to temperature.egg-info/top_level.txt
writing dependency_links to temperature.egg-info/dependency_links.txt
reading manifest file 'temperature.egg-info/SOURCES.txt'
writing manifest file 'temperature.egg-info/SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.rst,
README.txt

running check
creating temperature-1.0.0
creating temperature-1.0.0/temperature.egg-info
making hard links in temperature-1.0.0...
hard linking setup.py -> temperature-1.0.0
hard linking temperature.py -> temperature-1.0.0
hard linking temperature.egg-info/PKG-INFO -> temperature-1.0.0/temperature.egg-info
hard linking temperature.egg-info/SOURCES.txt -> temperature-1.0.0/temperature.egg-info
hard linking temperature.egg-info/dependency_links.txt -> temperature-
1.0.0/temperature.egg-info
hard linking temperature.egg-info/top_level.txt -> temperature-1.0.0/temperature.egg-info
Writing temperature-1.0.0/setup.cfg
Creating tar archive
removing 'temperature-1.0.0' (and everything under it)
```

```
tree dist
dist
└── temperature-1.2.tar.gz
```

Creating wheels

- 3 kinds of wheels
  - Universal wheels (pure Python; python 2 *and* 3 compatible
  - Pure Python wheels (pure Python; Python 2 *or* 3 compatible
  - Platform wheels (Platform-specific; binary)

A wheel is prebuilt distribution. Wheels can be installed with pip.

A Universal wheel is a pure Python package (no extensions) that can be installed on either Python 2 or Python 3. It has to have been carefully written that way.

A Pure Python wheel is a pure Python package that is specific to one version of Python (either 2 or 3). It can only be installed by a matching version of pip.

A Platform wheel is a package that has extensions, and thus is platform-specific.

Example

```
python setup.py bdist_wheel
running bdist_wheel
running build
running build_py
creating build
creating build/lib
copying temperature.py -> build/lib
installing to build/bdist.macosx-10.6-x86_64/wheel
running install
running install_lib
creating build/bdist.macosx-10.6-x86_64
creating build/bdist.macosx-10.6-x86_64/wheel
copying build/lib/temperature.py -> build/bdist.macosx-10.6-x86_64/wheel
running install_egg_info
running egg_info
writing temperature.egg-info/PKG-INFO
writing top-level names to temperature.egg-info/top_level.txt
writing dependency_links to temperature.egg-info/dependency_links.txt
reading manifest file 'temperature.egg-info/SOURCES.txt'
writing manifest file 'temperature.egg-info/SOURCES.txt'
Copying temperature.egg-info to build/bdist.macosx-10.6-x86_64/wheel/temperature-1.0.0
-py2.7.egg-info
running install_scripts
creating build/bdist.macosx-10.6-x86_64/wheel/temperature-1.0.0.dist-info/WHEEL
```

```
tree dist
dist
├── temperature-1.2-py3-none-any.whl
└── temperature-1.2.tar.gz
```

Creating other built distributions

- Use setup.py with -bdist –format=format
- Creates platform-specific distributions
- Common Unix formats: rpm, deb, tgz
- Common Windows formats: msi, exe

For the convenience of the end-user, you can create "built" distributions, which are ready to install on specific platforms. These are built with the bdist argument to setup.py, plus a --format=format option to indicate the target platform.

```
python setup.py bdist --format=rpm
```

```
tree dist
dist
├── temperature-1.2-py3-none-any.whl
├── temperature-1.2.macosx-10.9-x86_64.tar
├── temperature-1.2.macosx-10.9-x86_64.zip
└── temperature-1.2.tar.gz
```

Installing a package

- Use pip for wheels
- Use setup.py for source

One of the advantages of wheels is that they make installing packages easier. You can just use

pip install *package*.whl

If you have a source distribution, extract the source in any convenient location (this is not permanent) and cd to the top-level folder. Use the following command:

```
python setup.py install
```

To install in the default location.

Use

```
python setup.py install --prefix=ALTERNATE-DIR
```

to install under an alternate prefix.

Using Cookiecutter

- Create standard layout
- Developed for Django
- Very flexible

cookiecutter is a utility written by Audrey and Roy Greenfeld to make it easy to replicate a standard setup for Django. The cookiecutter command prompts you for information, then creates the project folder.

It uses a cookiecutter *template*, which is a folder, to create the new project. There are many templates on **github** to choose from, and you can easily create your own.

The script copies the template layout (all folders and files) to a new folder which is the "slug" (short name) of your project. It inserts your project name in the appropriate places.

cookiecutter home page: https://github.com/audreyr/cookiecutter
cookiecutter docs: https://cookiecutter.readthedocs.io

Package files the new way

- Create `pyproject.toml`
- `setup.py` and `setup.cfg` not needed
  - (probably)
- Use `build` to build the package

The **modern** way to build a Python package is using the `pyproject.toml` config file. The specifications that support this are specified in **PEP 518** and **PEP 621**.

The **TOML** format is similar to `.ini` files, but adds some features.

The first part of the file is required. It tells the `build` program what tools to use.

```
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"
```

the rest of the file is not needed if you are also using `setup.cfg` and `setup.py`. However, there is a trend away from using those legacy files, as all the data needed for building the package, installing it, and uploading it to **PyPI** can be contained in `pyproject.toml`, and can be used by any build *backend*.

```
[project]
name = "wombatfun"
version = "1.0.0"
authors = [
    { name="Author Name", email="jstrickler@gmail.com" },
]
description = "Short Description of the Package"
readme = "README.rst"
requires-python = ">=3.0"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]

dependencies = [
    'requests[security] < 3',
]
```

TIP In TOML, value types arrays are similar to Python `list`s and tables (including inline)

> are similar to `dict`.

Building the project

- `python -m build`
- Creates `dist` folder
- Binary distribution
  - `package-version.whl`
- Source distribution
  - `package-version.tar.gz`

To build the project, use

```
python -m build
```

This will create the wheel file (binary distribution) and a gzipped tar file (source distribution) in a folder named `dist`.

Editable installs

- Use `pip install -e package`
- Puts a link in library folder
- Allows testing as though module is installed

When using a `src` (or other name) folder for your codebase and `tests` for your test scripts, the tests need to find your package. While you could put the path to the `src` folder in PYTHONPATH, the best practice is to do an *editable install*.

This is an install that uses the path to your development folder. It achieves this by using a virtual environment. Then you can run your tests after making changes to your code, without having to reinstall the package.

From the top level folder of the project, type the following (you do not have to build the distribution for this step).

```
pip install -e .
```

Then you can just say

```
pytest -v
```

to run the tests in verbose mode, or any other variation of `pytest`.

Combining setup and build

- Provide backward compatibility
- TOML approach can have glitches

to be completed

For more information

Python Packaging User Guide

https://packaging.python.org/en/latest/

Distributing Python Modules

https://docs.python.org/3/distributing/index.html

setuptools Quickstart

https://setuptools.pypa.io/en/latest/userguide/quickstart.html

Thoughts on the Python packaging ecosystem

https://pradyunsg.me/blog/2023/01/21/thoughts-on-python-packaging/

THE BASICS OF PYTHON PACKAGING IN EARLY 2023

https://drivendata.co/blog/python-packaging-2023

Structuring Your Project (from The Hitchhiker's Guide to Python)

https://docs.python-guide.org/writing/structure/

Chapter 1 Exercises

Exercise 1-1 (president/*)

Implement a distributable package from the **President** class created in the chapter on object-oriented programming.

Create a wheel file, then try to install it with **pip**.

Appendix A: Where do I go from here?

Resources for learning Python

These are from Jessica Garson, who, among other things, teaches Python classes at NYU. (Used with permission).

Run the script **where_do_i_go.py** to display a web page with live links.

Resources for Learning Python

Just getting started

Here are some resources that can help you get started learning how to code.

- Code Newbie Podcast
- Dive into Python3
- Learn Python the Hard Way
- Learn Python the Hard Way
- Automate the Boring Stuff with Python
- Automate the Boring Stuff with Python

So you want to be a data scientist?

- Data Wrangling with Python
- Data Analysis in Python
- Titanic: Machine Learning from Disaster
- Deep Learning with Python
- How to do X with Python
- Machine Learning: A Probabilistic Prospective

So you want to write code for the web?

- Learn flask, some great resources are listed here
- Django Polls Tutorial
- Hello Web App
- Hello Web App Intermediate
- Test-Driven-Development for Web Programming
- 2 Scoops of Django

- HTML and CSS: Design and Build Websites
- JavaScript and JQuery

Not sure yet, that's okay!

Here are some resources for self guided learning. I recommend trying to be very good at Python and the rest should figure itself out in time.

- Python 3 Crash Course
- Base CS Podcast
- Writing Idiomatic Python
- Fluent Python
- Pro Python
- Refactoring
- Clean Code
- Write music with Python, since that's my favorite way to learn a new language

Appendix B: Python Bibliography

Data Science

- ***Building machine learning systems with Python***. William Richert, Luis Pedro Coelho. Packt Publishing
- ***High Performance Python***. Mischa Gorlelick and Ian Ozsvald. O'Reilly Media
- ***Introduction to Machine Learning with Python***. Sarah Guido. O'Reilly & Assoc.
- ***iPython Interactive Computing and Visualization Cookbook***. Cyril Rossant. Packt Publishing
- ***Learning iPython for Interactive Computing and Visualization***. Cyril Rossant. Packt Publishing
- ***Learning Pandas***. Michael Heydt. Packt Publishing
- ***Learning scikit-learn: Machine Learning in Python***. Raúl Garreta, Guillermo Moncecchi. Packt Publishing
- ***Mastering Machine Learning with Scikit-learn***. Gavin Hackeling. Packt Publishing
- ***Matplotlib for Python Developers***.Sandro Tosi.Packt Publishing
- ***Numpy Beginner's Guide.Ivan Idris***.Packt Publishing
- ***Numpy Cookbook.Ivan Idris***.Packt Publishing
- ***Practical Data Science Cookbook.Tony Ojeda, Sean Patrick Murphy, Benjamin Bengfort, Abhijit Dasgupta***.Packt Publishing
- ***Python Text Processing with NLTK 2.0 Cookbook.Jacob Perkins***.Packt Publishing
- ***Scikit-learn cookbook.Trent Hauck***.Packt Publishing
- ***Python Data Visualization Cookbook.Igor Milovanovic***.Packt Publishing
- ***Python for Data Analysis.Wes McKinney.***. O'Reilly & Assoc

Design Patterns

- ***Design Patterns: Elements of Reusable Object-Oriented Software.Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides***.Addison-Wesley Professional
- ***Head First Design Patterns.Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra***.O'Reilly Media
- ***Learning Python Design Patterns.Gennadiy Zlobin***.Packt Publishing
- ***Mastering Python Design Patterns.Sakis Kasampalis***.Packt Publishing

General Python development

- ***Expert Python Programming.Tarek Ziadé***.Packt Publishing
- ***Fluent Python.Luciano Ramalho***. O'Reilly & Assoc.

- ***Learning Python, 2nd Ed..Mark Lutz, David Asher***. O'Reilly & Assoc.
- ***Mastering Object-oriented Python.Stephen F. Lott***.Packt Publishing
- ***Programming Python, 2nd Ed. .Mark Lutz***. O'Reilly & Assoc.
- ***Python 3 Object Oriented Programming.Dusty Phillips***.Packt Publishing
- ***Python Cookbook, 3rd. Ed.. David Beazley, Brian K. Jones***. O'Reilly & Assoc.
- ***Python Essential Reference, 4th. Ed..David M. Beazley***.Addison-Wesley Professional
- ***Python in a Nutshell.Alex Martelli***. O'Reilly & Assoc.
- ***Python Programming on Win32.Mark Hammond, Andy Robinson***. O'Reilly & Assoc.
- ***The Python Standard Library By Example.Doug Hellmann***.Addison-Wesley Professional

Misc

- ***Python Geospatial Development.Erik Westra***.Packt Publishing
- ***Python High Performance Programming.Gabriele Lanaro***.Packt Publishing

Networking

- ***Python Network Programming Cookbook.Dr. M. O. Faruque Sarker***.Packt Publishing
- ***Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers.T J O'Connor***.Syngress
- ***Web Scraping with Python.Ryan Mitchell***.O'Reilly & Assoc.

Testing

- ***Python Testing Cookbook.Greg L. Turnquist***.Packt Publishing
- ***Learning Python Testing.Daniel Arbuckle***.Packt Publishing
- ***Learning Selenium Testing Tools, 3rd Ed. .Raghavendra Prasad MG***.Packt Publishing

Web Development

- ***Building Web Applications with Flask.Italo Maia***.Packt Publishing
- ***Django 1.0 Website Development.Ayman Hourieh***.Packt Publishing
- ***Django 1.1 Testing and Development.Karen M. Tracey***.Packt Publishing
- ***Django By Example.Antonio Melé***.Packt Publishing
- ***Django Design Patterns and Best Practices.Arun Ravindran***.Packt Publishing
- ***Django Essentials.Samuel Dauzon***.Packt Publishing

- ***Django Project Blueprints.Asad Jibran Ahmed***.Packt Publishing
- ***Flask Blueprints.Joel Perras***.Packt Publishing
- ***Flask by Example.Gareth Dwyer***.Packt Publishing
- ***Flask Framework Cookbook.Shalabh Aggarwal***.Packt Publishing
- ***Flask Web Development.Miguel Grinberg***. O'Reilly & Assoc.
- ***Full Stack Python (e-book only).Matt Makai***.Gumroad (or free download)
- ***Full Stack Python Guide to Deployments (e-book only).Matt Makai***.Gumroad (or free download)
- ***High Performance Django.Peter Baumgartner, Yann Malet***.Lincoln Loop
- ***Instant Flask Web Development.Ron DuPlain***.Packt Publishing
- ***Learning Flask Framework.Matt Copperwaite, Charles O Leifer***.Packt Publishing
- ***Mastering Flask.Jack Stouffer***.Packt Publishing
- ***Two Scoops of Django: Best Practices for Django 1.11. Daniel Roy Greenfeld, Audrey Roy Greenfeld***.Two Scoops Press
- ***Web Development with Django Cookbook.Aidas Bendoraitis***.Packt Publishing

Index