Python Packaging

John Strickler

Version 1.0, February 2021

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
- Distribute and deploy packages

How to package

- Copy app to new folder
- Add standard files
- Use setuptools to build package

To create a Python package, add standard files, as described next to the top level of the project.

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

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

Package files

- 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
# from setuptools import find_packages

setup(
    name='temperature',
    version='1.2',
    description='Class for converting temperatures',
    long_description='Long description here....',
    author='John Strickler',
    author_email='jstrickler@gmail.com',
    license='MIT',
    url='http://www.cja-tech.com/temperature',  # doesn't really exist
    py_modules=['temperature'],
#    packages=find_packages(exclude=['contrib', 'doc', 'tests']),
)
```

```
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
```

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

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.

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**.

Index