Python Packaging Basics

Alan Pearl

AstroPGH Summer Seminar Series

July 13, 2022

Why create your own Python package?

- It's easy
  - Simple format, easily install with `pip install .`
- Simplify your Jupyter notebooks by hiding your messy code
  - No more copy-and-pasting all of your code from notebook to notebook
- Share your code with collaborators
  - With a quick README file, anyone in the world can install your package from GitHub

Vocabulary - the components of a package

- A *module* is a Python file that you intend to import
- A *script* is a Python file that you intend to execute
- A *package* is a directory that must contain an `__init__.py` module
- Besides `__init__.py`, a package is allowed to contain other modules, scripts, and sub-packages
- **Importing** a package simply imports the `__init__.py` module
- **Executing** a package executes the `__main__.py` script, if it exists

Example structure

- This example package is called pypackbasics.
- Note that setup.py and doc files go *outside* the package, in the root
- Package and module names should be concise, lower-case, with no dashes, colons, etc.
- Underscores are okay if necessary for readability

```
+ pypackbasics  (root directory)
|   - setup.py
|   - README.md
|   + pypackbasics  (package)
|   |   - __init__.py  (module)
|   |   - basics.py  (module)
|   |   + utilpack  (package)
|   |   |   - __init__.py  (module)
|   |   |   - utils.py  (module)
|   |   + otherpack  (package)
|   |   |   - __init__.py  (module)
|   |   |   - __main__.py  (script)
|   |   |   - other.py  (module)
|   |   |   - another.py  (module)
```

Install it yourself

- View this example repository at https://github.com/AlanPearl/pypackbasics
- You can fork this repo so you can always use it as a template for yourself
- Let's look through and edit this package together. Clone it to your computer with: `git clone https://github.com/<user>/pypackbasics`
- Follow the pip install instructions

```
+ pypackbasics  (root directory)
|   - setup.py
|   - README.md
|   + pypackbasics  (package)
|   |   - __init__.py  (module)
|   |   - basics.py  (module)
|   |   + utilpack  (package)
|   |   |   - __init__.py  (module)
|   |   |   - utils.py  (module)
|   |   + otherpack  (package)
|   |   |   - __init__.py  (module)
|   |   |   - __main__.py  (script)
|   |   |   - other.py  (module)
|   |   |   - another.py  (module)
```

How does pip know how to install it?

- The package is specified by setup.py using the find_packages function (see below)

```
from setuptools import setup, find_packages


setup(
    name="pypackbasics",
    version="1.0",
    description="Python Packaging Basics: An educational template
package",
    url="https://github.com/AlanPearl/pypackbasics",
    author="Alan Pearl",
    author_email="alanpearl@pitt.edu",
    license="MIT",
    python_requires=">=3.6",  # note: 3.6 is required for f-strings
    install_requires=[
        "matplotlib",
        "numpy>=1.18",
    ],
    packages=find_packages()
)
```

```
+ pypackbasics  (root directory)
|    - setup.py
|    - README.md
|    + pypackbasics  (package)
|    |    - __init__.py  (module)
|    |    - basics.py  (module)
|    |    + utilpack  (package)
|    |    |    - __init__.py  (module)
|    |    |    - utils.py  (module)
|    |    + otherpack  (package)
|    |    |    - __init__.py  (module)
|    |    |    - __main__.py  (script)
|    |    |    - other.py  (module)
|    |    |    - another.py  (module)
```

Importing this package

- Open a python console or notebook anywhere *outside* of the pypackbasics directory
- You can now simply `import pypackbasics` to access most of the classes and functions
- However, `__init__.py` doesn't import any code from the `otherpack` package, so you will need to explicitly `import pypackbasics.otherpack`
- The executable script can be run via `python -m pypackbasics.otherpack`

```
+ pypackbasics  (root directory)
|    - setup.py
|    - README.md
|    + pypackbasics  (package)
|    |    - __init__.py  (module)
|    |    - basics.py  (module)
|    |    + utilpack  (package)
|    |    |    - __init__.py  (module)
|    |    |    - utils.py  (module)
|    |    + otherpack  (package)
|    |    |    - __init__.py  (module)
|    |    |    - __main__.py  (script)
|    |    |    - other.py  (module)
|    |    |    - another.py  (module)
```

Unit tests

- Adding unit tests saves time in the long run
- As you develop your code, you don’t want to break functionality that was previously working
- Example below (test_basics.py)

```
import unittest

import pypackbasics


class TestPrimeFinder(unittest.TestCase):
    def test_find_primes_up_to_10(self):
        finder = pypackbasics.PrimeFinder()
        finder.find_primes(max_prime=10)
        assert finder.known_primes == [2, 3, 5, 7]
        assert finder.highest_check == 10
```

```
+ pypackbasics  (root directory)
|    - setup.py
|    - README.md
|    + pypackbasics  (package)
|    |    - __init__.py  (module)
|    |    - basics.py  (module)
|    |    + utilpack  (package)
|    |    |    - __init__.py  (module)
|    |    |    - utils.py  (module)
|    |    + otherpack  (package)
|    |    |    - __init__.py  (module)
|    |    |    - __main__.py  (script)
|    |    |    - other.py  (module)
|    |    |    - another.py  (module)
|    + tests/  (package)
|    |    - __init__.py  (module)
|    |    -test_basics.py  (module)
```