Reproducible Research with Python

Software Development for Scientists

James Bourbeau

December 5, 2017

GitHub repository https://github.com/jrbourbeau/xmeeting-reproducible-research

Comments on reproducibility

- Same inputs give the same output.
- Code can easily run by someone else.
- Environment needed to run code can be created easily.

Topics

- Python packages
- Making packages `pip`-installable
- Virtual environments
- Writing tests with `pytest`

Python packages

Python packages

- A Python package is a directory containing:
  1. `.py` files containing Python code (called "modules")
  2. `__init__.py` file.
- I.e. a package is a collection of modules.
- Can act as a useful organizational tool to centralize code.
- We'll talk more about `__init__.py` later...

Example Python package

- For example, a Python package could look like

```
expackage/
    |- __init__.py
    |- math.py
```

```python
# math.py
def my_add(a, b):
    return a + b
```

Importing a package

- While in the same directory as a Python package, you can `import` and use the code inside package modules.
- Dot syntax (`<package>.<module>`) is used to access the module namespace.

```
>>> from expackage.math import my_add
>>> my_add(2, 3)
5
```

So what's `__init__.py` for?

- The contents of `expackage/__init__.py` are run whenever `expackage` is imported.
- Any code can be placed in `__init__.py`.
- Often used to lift code from the module-level namespace up into the top-level package namespace (example on next slide).

Lifting module code into package namespace

- Import code from modules inside `__init__.py`.

```
# __init__.py
from .math import my_add
```

- Now available at the top-level namespace.

```
>>> import expackage
>>> expackage.my_add(2, 3)
5
```

`import`-ing troubles

- Will run into issues when you're not in the same directory as `expackage`.

```
>>> import expackage
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'expackage'
```

- How does Python find packages?

How does Python find packages?

- Question: How does Python find packages?
- Answer: It looks in `sys.path`

```
>>> import sys
>>> sys.path
['',
 '/usr/local/lib/python36.zip',
 '/usr/local/lib/python3.6',
 '/usr/local/lib/python3.6/plat-darwin',
 '/usr/local/lib/python3.6/lib-dynload',
 '/usr/local/lib/python3.6/site-packages']
```

How is `sys.path` constructed?

- `sys.path` includes:
  1. The current working directory.
  2. Anything specified in the `PYTHONPATH` environment variable.
  3. Site-specific paths (`site.py` module)
- I don't recommend messing with the `PYTHONPATH` variable unless you absolutely have to.
  - See PYTHONPATH Considered Harmful blog post.

Installing Python packages

- `pip` is the recommended Python package installer.
- Installs packages into `site-packages/` directory by default.
- Easy to use `pip` (e.g. `pip install numpy`).
- `pip` knows about versions.
  - Latest version: `pip install numpy`
  - Specific version: `pip install numpy==1.13.0`
  - Minimum version: `pip install numpy>=1.11.1`
- Also works with version control systems.
  - `pip install git+https://github.com/numpy/numpy.git#egg=numpy`

Requirements files

- Requirements files are files containing a list of items to be installed using `pip install`.
- `pip install -r requirements.txt`
- Effectively, each line in a requirements file gets passed to `pip install`.

```
# requirements.txt
numpy==1.11.0
matplotlib
scikit-learn>=0.18.2
```

Making packages `pip`-installable

Making a Python package installable with `pip`

- To make your own Python package installable with `pip`, you need a `setup.py` file.

```
# setup.py
from setuptools import setup

setup(
    name='expackage',
    description='Example Python package',
    packages=['expackage'],
)
```

- `pip` and `setuptools` are included with Python 2 >= 2.7.9 and Python 3 >= 3.4.

Specifying package dependencies

```
# setup.py
from setuptools import setup

setup(
    name='expackage',
    description='Example Python package',
    packages=['expackage'],
    install_requires=['numpy==1.13.0',
                      'matplotlib',
                      'scikit-learn>=0.18.2']
)
```

Standard package layout

- Typically, there's a top level directory that contains the `setup.py` file, the actual Python package, and any additional files (e.g. `README`, license, etc).

```
expackage/
    setup.py
    README.md
    expackage/
        |- __init__.py
        |- math.py
```

- It's convention to have the package name and top level directory name to match, but it's not required.
- Not quite ready to `pip install expackage` yet.

Virtual environments

Why use virtual environments?

- Different projects have different dependencies.
  - Project A needs `pandas==0.18.0`.
  - Project B needs `pandas==0.16.0`.
- You can't install packages into the global `site-packages/` directory. For instance, on a shared host.

What is a virtual environment?

- Isolated Python environment with its own separate version of `pip` that installs packages into its own `site-packages/` directory.
- Virtual environments don't talk to each other.
- Each project you work on can have its own Python packages installed separately.
  - Project A can have `pandas==0.18.0`
  - Project B can have `pandas==0.16.0`.

Mental picture of virtual environments

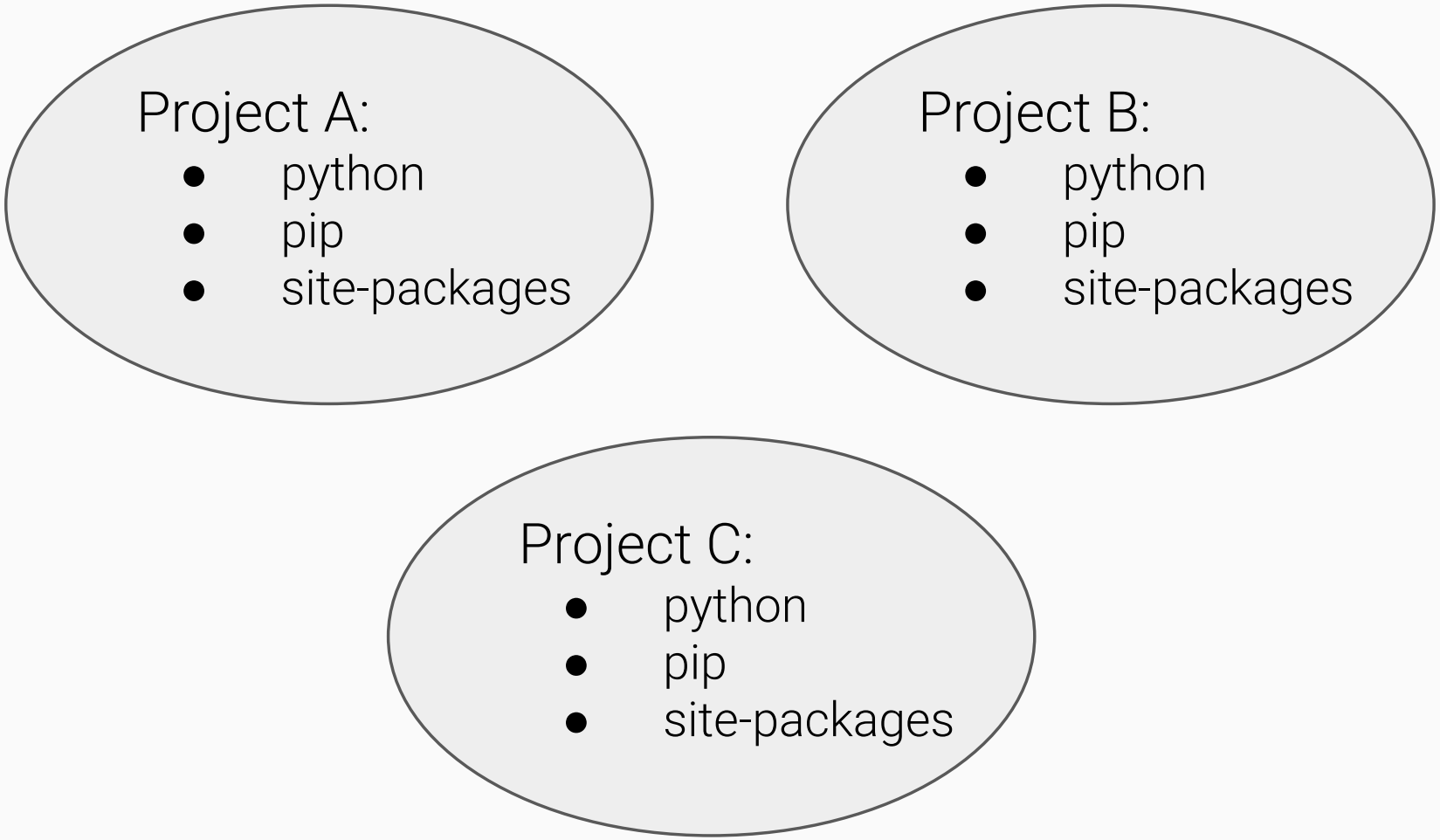

Figure 1: Virtual environments are isolated from one another. Packages installed into Project A's virtual environment can't be used in Project B, and vice versa.

How to create a virtual environment

- Depends on which version of Python you are using.
- Python 3.3+ has a `venv` module build into the standard library.

```
$ python3 -m venv ~/.venvs/project-a
$ source ~/.venvs/project-a/bin/activate
(project-a) $
```

- Python 2.6+ and Python 3.3+ should use the third party `virtualenv` package (open source and maintained by the Python Packaging Authority).

```
$ pip install virtualenv
$ virtualenv ~/.venvs/project-a
$ source ~/.venvs/project-a/bin/activate
(project-a) $
```

They really are isolated!

```
$ which python
/usr/local/bin/python

$ source ~/.venvs/project-a/bin/activate

(project-a) $ which python
/Users/jbourbeau/.venvs/project-a/bin/python

(project-a) $ which pip
/Users/jbourbeau/.venvs/project-a/bin/pip
```

Writing tests with `pytest`

What are tests?

- Software tests are checks that test the quality of your code.
  - Unit tests — test that a specific function is working as expected.
  - Usability tests — test that code interacts as expected with the user.
- Tests are useful to make sure that your code works as expected. Also ensures that future changes made to your code don't break it in unexpected ways.

Testing tools

- There is a `unittest` module built into the Python standard library.
- `pytest` is a framework for writing tests for Python code.
  - Easy to install: `pip install pytest`.
  - Light-weight syntax for writing tests.
  - Large plugin community.
  - Supports running `unittest` tests.

Writing our first test

```
expackage/
    setup.py
    README.md
    expackage/
        |- __init__.py
        |- math.py
        |- tests/
            |- test_math.py
```

```python
# test_math.py
from expackage import my_add

def test_my_add():
    assert my_add(3, 2) == 5
```

Running tests with `pytest`

- Use the `pytest` command to run your tests.
- Starts in specified directory (current working directory otherwise) and looks for `test_*.py` or `*_test.py` files.
  - `pytest expackage` will run all tests in `expackage/*/test_*.py` or `expackage/*/*_test.py` files.
- `pytest` has many useful command line options.
  - `pytest -sv expackage`

Adding some input verification

```
def my_add(a, b):

    if not isinstance(a, (int, float)):
        raise TypeError('Input to my_add should be '
                        'either integers or floats')

    return a + b
```

Testing for failure

```
# test_math.py
import pytest
from expackage import my_add

def test_my_add():
    assert my_add(3, 2) == 5

def test_my_add_first_input_raises():
    with pytest.raises(TypeError) as excinfo:
        my_add('not a number', 5)
    error = ('Input to my_add should be either '
             'integers or floats')
    assert error == str(excinfo.value)
```

Summary

- Structure of Python packages.
- Making packages installable with `pip`.
- Virtual environments.
  - What are they?
  - How do I create one?
- Writing tests with `pytest`

Other stuff to check out

- `virtualenvwrapper` — set of extensions to `virtualenv`. It will save you keystrokes!
- Travis CI for automatically running your tests (free for any public GitHub repositories).
- `conda` — open source package management system and environment management system.

Additional resources

- David Beazley's "Modules and Packages: Live and Let Die!" talk @ PyCon 2015 [youtube] [slides]
- David Baumgold's "Get Started With Git" talk @ PyCon 2016 [youtube] [slides]
- How To Package Your Python Code tutorial http://python-packaging.readthedocs.io