Develpment Practices, Modules, Tools & Releases.

ENGF0002: Design and Professional Skills

Mark Handley
based on slides by George Danezis

Term 1, 2019

UCL

Python packages, libraries, and deployment.

Import Python libraries

Import a library by name:

```
>>> import statistics
>>> statistics.median([1, 2, 3, 4, 8])
3
```

Import specific names within the library:

```
>>> from statistics import median, mean
>>> median([1, 2, 3, 4, 8])
```

Import and rename a library:

```
>>> import statistics as stat
>>> stat.median([1, 2, 3, 4, 8])
```

Look for library in the folders `sys.path`.

The Python Standard Library

The standard libraries are available with any full python installation. You can **rely on them**!

- **Basic programming**: Built-in types, text processing, binary data, dates, calendars, maths, functional tools, . . .
- **System**: Files, directories, data formats, compression, cryptography, operating system functions, . . .
- **Communications**: Inter-process communications, internet protocols, markup, multimedia, i19n, . . .
- **Language**: Debug / profile, distrbute, runtime, interpreters, import, language, . . .

Always **prefer to use the standard library** over your own types.

Non-standard libraries

Third party libraries require **additional installation**. **Useful ones**:

numpy,scipy Numeric & scientific libraries and linear algebra.

matplotlib Scientific graphs and plots.

pandas Manipulation of tabular data.

OpenCV,scikit-learn,Pytorch,Theano,TensorFlow Machine learning.

tox, pytest, sphinx Tests, documentation.

requests,django,flask Web requests and web app servers.

fabric,doit DevOps.

cffi Bind to low-level C code.

Should you take a dependency?

Issues to consider before relying on a 3rd party library:

- **Quality**: is it going to be any good? Good Documentation; No Bugs.
- **Community**: is it being used by many? is it being maintained? Release schedule, responsiveness to bug reports, and breaking downstreams.
- **Trust**: do you believe maintainers will attack your software?
- **License**? are you allowed to use it?

If you are the biggest user of a library by far, you may end up having to also be the maintainer.

Understanding Intellectual Property

The state is protecting creators to incentivise investment in new inventions.

- **Copyright**: protects authors of artistic or literary works, including software! if you write something you (or employer) own the copyright – can transfer it, or license it.
- **Trade marks**: protect your trading image, name, logos, etc. to avoid brand and consumer confusion. You might have to register them.
- **Patents**: apply and get issued one for an invention. Disclose the invention, but get protection for a limited time. Status of software patents is problematic.

Licensing your own software, or using licensed software.

A software license is an agreement between a creator and a user of a work, about the conditions under which the user may use the creations.

- **Commercial**: creator gets some money, and in return allows use of software. Usually no other rights are transfered.
- **Free Software**: creator allows user all rights, including modifying software. But does not allow endorsement, or liability. May or not grant **rights to patents**.
  eg. BSD, MIT, Apache.
- **Gnu Public Licenses**: like free software, but **viral**. Requires any modifications to also become free software under the same license.
  eg. GPLv2, GPLv3, LGPL.

Check license before using libraries!

» About
» Features
» Applications
» Advocacy
» Marketing
» Administration
» News
» Events
» Press
» Multimedia
» Artwork
» Logo
» Donations
» Legal Notices
» Privacy Policy

The FreeBSD Copyright

Full License Text

Edit

GNU General Public License, version 2 (GPL-2.0)

The GNU General Public License (GPL-2.0)

Version 2, June 1991

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies

of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You

Philosophical discussion: what is more ‘free’?

Question 1. Viral licenses impose more restrictions on users, but guarantee the resulting software is also free. Are they more or less in line with ideas of software freedom?

Question 2. Discuss the statement ‘A good non-viral library will inevitably end up being extended commercially and become closed source / non-free as a result.’

Question 3. What incentives are there for commercial entities to contribute to free software?

Also, study controversies around software patents:
`http://endsoftpatents.org/`

Installing 3rd party modules

List of great Python modules:
`https://github.com/vinta/awesome-python`
Python Package repository, **Pypi**: `https://pypi.python.org/pypi`

Python comes with `pip` as a package manager:

```
$ pip install <modulename>
```

Full `pip` documentation: `https://pip.pypa.io/en/stable/`

Clashing versions of modules

What to do if you need two different (clashing) versions of the same library?
What about testing your program or library using different library versions?

Solution: use `virtualenv` to create a clean virtual environment:

```
$ virtualenv ENV              # Create in dir ENV
$ source ENV/bin/activate     # Activate environment
(ENV)$ pip install ...        # Install in ENV
(ENV)$ python ...             # Run in ENV
(ENV)$ deactivate             # Exit ENV
$
```

Make your own libraries and modules

Why think of programs in terms of modules?

- Most programs are too big to live in a single file.
- Clean, well defined interfaces between modules, reduce communication overheads.
- Information Hiding behind modules allows for abstraction and maintainability.
- Controlled interactions allow decoupling of modules.
- High-quality, reusable code – can be used across projects.

Lightweight libraries and modules

Python allows a program to be **split across many files**.

Consider the two files:

`branch.py`
`minimap.py`

Each creates a **name space available to the other** through `import`.
Eg. in `minimap.py`:

```
from branch import branch
```

From many files, to libraries, to modules . . .

The structure of a module

All user-facing libraries, and Python end-user programs are usually packaged as **modules**.

Module `<mymodule>` is a directory of files:

```
README.rst                  # Basic help on github
LICENSE                     # License
setup.py                    # Packaging & dependencies information
tox.ini                     # Test setup
euclid/__init__.py          # Modules code
euclid/gcd.py
docs/conf.py                # Module Sphinx documentation
docs/index.rst
tests/test_gcd.py           # Module tests
```

See The Hitchhiker's Guide to Python

`http://docs.python-guide.org/en/latest/`

The GCD function – as before

We define our GCD function in `euclid/gcd.py`

```
    def GCD(a,b):
        """Compute the GCD of two positive integers.

        >>> GCD(10,5)
        5

        """
        if not (a > 0 and b > 0):
            raise ArithmeticError("%s, %s: Must be positive int." % (a,b))
        while (b != 0):
            a, b = b, a % b
        return a
```

The module `__init__.py` file

A `euclid/__init__.py` file indicates the directory defines a module.

The code in the file defines names within the module.

The variable `__all__` restricts the names exported,
by eg. `from euclid import *`.

The module `__init__.py` file

```
__all__ = [ "GCD", "main" ] # names exported by module

from .gcd import GCD          # relative import within module
import sys

def main():
    try: # Take the two first command-line args, as integers.
        ax = int(sys.argv[1])
        bx = int(sys.argv[2])
        print(GCD(ax, bx))
    except IndexError:
        print("Euclid requires two arguments.")
    except ValueError:
        print("Euclid requires two integers.")
    except ArithmeticError:
        print("Euclid requires positive integers.")

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

The module `setup.py`

```
# Always prefer setuptools over distutils
from setuptools import setup

setup(
    name='euclid',      version='0.0.1',
    description='Euclid greatest common denominator (GCD) library',
    long_description="....",
    url='https://github.com/mhandley/ENGF0002',
    author='George Danezis, University College London',
    author_email='g.danezis@ucl.ac.uk',
    license='BSD',

    packages=['euclid'],            # The package directories
    install_requires=['pytest==3.3.0'],  # Dependencies

    entry_points={                  # Entry function
        'console_scripts': ['euclid=euclid:main'] },
)
```

The `setup.py` meta-data

- The **name** and **version** are used by pip / pypi to index your package.
- The **packages** point to what is included in the module. (Note that tests are not included in this case.)
- The **install_requires** lists packages required. Can specify **version range**.
- The **entry_points** list the function that starts an application.

Packaging the module

Create a **source distribution** using:

```
$ python setup.py sdist
```

Create a **binary distribution** using:

```
$ python setup.py bdist
$ python setup.py bdist --format=wininst
```

Install manually a package:

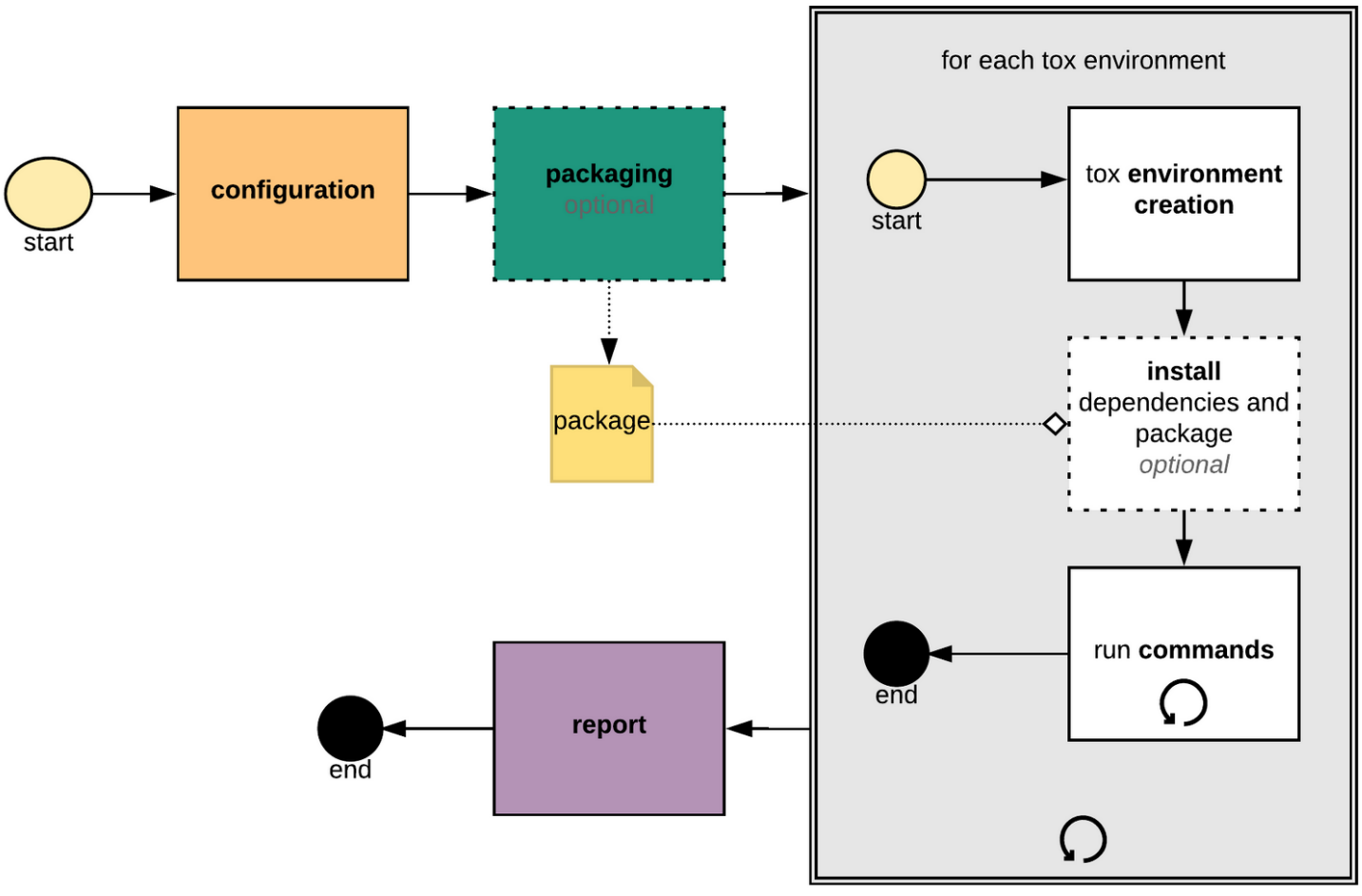
```
$ python setup.py install
```

Testing packaging and installation using `tox`

- Need to test packaging.
- Need to test install process.
- Need to test against many Python versions.
- Need to test against many library versions.

The tool of choice for this is `tox`. You can define **environments**, **dependencies**, and **commands** to test in them

Tox workflow



The `tox.ini` file

Test in a Python 3.5 context . . .

```
[tox]
envlist = py35

[testenv]
deps = pytest
       pytest-cov

commands = pytest --doctest-modules -sv euclid
           pytest --cov={envsitepackagesdir}/euclid --doctest-modules -sv tests
           euclid 10 2
```

. . . and execute pytest, coverage and the program itself.

```
$ tox
GLOB sdist-make: src\euclid\setup.py
py35 inst-nodeps: src\euclid\.tox\dist\euclid-0.0.1.zip
py35 installed: coverage==4.4.2, euclid==0.0.1, pytest==3.3.0, pytest-cov==2.5.1
py35 runtests: commands[0] | pytest --doctest-modules -sv euclid
============================= test session starts =============================
collecting ... collected 1 item

euclid/gcd.py::euclid.gcd.GCD PASSED                                     [100%]
========================== 1 passed in 0.03 seconds ===========================
py35 runtests: commands[1] | pytest --cov=site-packages/euclid --doctest-modules -sv tests
============================= test session starts =============================
collecting ... collected 5 items

tests/test_gcd.py::test_euclid PASSED                                    [ 20%]
tests/test_gcd.py::test_euclid_exc PASSED                                [ 40%]
tests/test_gcd.py::test_euclid_exc_raises[test_inputs0] PASSED           [ 60%]
tests/test_gcd.py::test_euclid_exc_raises[test_inputs1] PASSED           [ 80%]
tests/test_gcd.py::test_euclid_exc_raises[test_inputs2] PASSED           [100%]
----------- coverage: platform win32, python 3.5.3-final-0 -----------
Name                                                 Stmts   Miss  Cover
------------------------------------------------------------------------
.tox\py35\Lib\site-packages\euclid\__init__.py          16     11    31%
.tox\py35\Lib\site-packages\euclid\gcd.py                6      0   100%
------------------------------------------------------------------------
TOTAL                                                   22     11    50%
========================== 5 passed in 0.06 seconds ===========================
py35 runtests: commands[2] | euclid 10 2
2
_________________________________ summary _________________________________
  py35: commands succeeded
  congratulations :)
```

Documentation

Use **Sphinx** and **autodoc** to generate documentation.

- Include (and test) examples in your documentation!
- Test your examples! (`pytest -doctest-modules -sv euclid`)
- Provide testing documentation – so that others can hack on your code.

Execute `sphinx-quickstart` to start your Sphinx documentation.

Documentation with `autodoc`

Write `index.rst`. Compile with `docs/make html`:

```
Welcome to euclid's documentation!
==================================

The euclid module provides an implementation of the Greatest Common Denominator (GCD) algorithm.
It provides a command line utility to compute the GCD::

    $ euclid 10 2
    2

The Euclid Python module
------------------------

.. autofunction:: euclid.GCD

Development
-----------

To test the Euclid module, use tox::

    $ tox
```

Quick search

Go

Welcome to euclid's documentation!

The euclid module provides an implementation of the Greatest Common Denominator (GCD) algorithm. It provides a command line utility to compute the GCD:

```
$ euclid 10 2
2
```

The Euclid Python module

euclid.GCD(*a*, *b*)

Compute the GCD of two positive integers.

```
>>> GCD(10,5)
5
```

Development

To test the Euclid module, use tox:

```
$ tox
```

Remember: Done Done!

The quality installation, packaging, documentation, distinguishes the great engineer:

- Always fold your projects into one or more modules.
- Provide packaging facilities through `setup` scrips.
- Allow installation through `pip`.
- Test in multiple environments using `tox`.
- Include `coverage` checking routinely in your QA process.
- Test packaging and deployment.
- Include and test documentation with `Sphinx`
- Upload into `pypi`, `travis`, `coveralls` and `readthedocs` ...

When learning other languages find the equivalent facilities.