First steps in packaging Python code

Konrad HINSEN

Centre de Biophysique Moléculaire (Orléans)

AND

Synchrotron Soleil (St Aubin)

Packaging

- Purpose: the preparation of a program or a library for distribution.

- Result: a single file that permits installation on any computer (ideally).

- Mandatory if you want to distribute your code, but very convenient just for yourself if you work on more than one computer.

- Good packaging leaves a good first impression of your program!

Questions to answer before starting

- For which platforms do I wish to package my program?
- What do I want to distribute? Source code? Executables? Documentation? Data?
- What are my external dependencies?

Your users will...

- look for a file called README and/or INSTALL for instructions

- prefer an installation procedure that he/she is already familiar with

- appreciate a short dependency list

Packaging and software development

- Ideally: two aspects of one process
- Use for development
  - the same file layout
  - the same tools (compilers etc.)

  that your clients will use for installation.

There is no clear borderline between development tools and packaging tools.

Packaging steps

1. Define an installation procedure that is simple to follow and as universal as possible.

2. Prepare documentation for your software.

3. Document the installation, not forgetting the dependency list.

4. Create a distribution file.

Packaging tools in the Python universe

Distribution and installation:

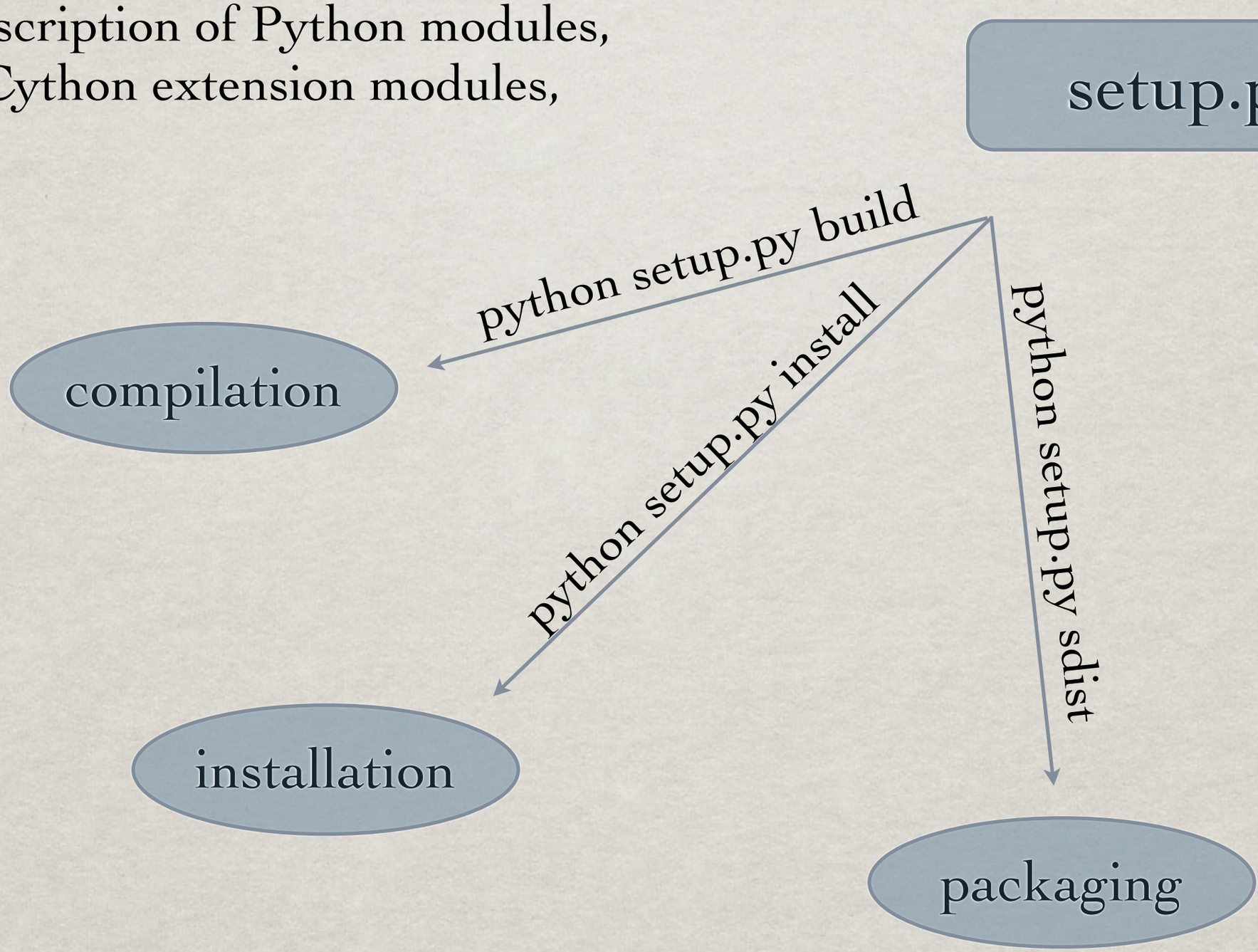
today → - distutils (in the Python standard library)
- NumPy distutils extensions (part of NumPy)
- Cython distutils extensions (part of Cython)
- setuptools / distribute (add dependency handling)
- bento (new alternative tool, under development)

Documentation:

- sphinx

Python distutils

Description of Python modules,
C/Cython extension modules,
etc.



http://docs.python.org/distutils/index.html

Minimal example

File sieve.py

```
def primes(n):
    """
    Returns the prime numbers between 1 and n-1,
    calculated using the Sieve of Eratosthenes.
    """
    numbers = range(2, n)
    primes = []
    while numbers:
        primes.append(numbers[0])
        numbers = [n for n in numbers[1:] if n % numbers[0] > 0]
    return primes
```

File setup.py

```
from distutils.core import setup
setup(name = "sieve",
        version = "0.1",
        author = "Eratosthenes",
        description = "Prime numbers",
        py_modules = ["sieve"],
        )
```

Exercise

Package your NumPy exercises

In the NumPy exercises, you will write a few functions working on arrays.

Put them into a module `array_utilities` and write a corresponding `setup.py`.