Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

MCS 507 Lecture 17
Mathematical, Statistical and Scientific Software
Jan Verschelde, 29 September 2023

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

a function definition

```
def fun(xarg, yarg, zarg):
    """
    Returns the value of the expression
      xarg**2*cos(yarg) + 4*exp(zarg)*sin(xarg)
    for numerical values of the formal arguments
    xarg, yarg, and zarg.
    """
    from math import exp, cos, sin
    result = xarg**2*cos(yarg) + 4*exp(zarg)*sin(xarg)
    return result
```

function definitions

A *function header* (e.g., `def fun(xarg, yarg, zarg):`) consists of

1. The name of the function (e.g., `fun`) follows `def`.
2. Formal arguments of the function are (e.g., `xarg`, `yarg`, `zarg`):
   - between round brackets `(` and `)`;
   - separated by commas.

   Round brackets are needed even if no arguments.
3. The colon `:` follows `)`.

The documentation string (between triple quotes) is optional, but is strongly recommended.

The *function body* may have local variables, e.g., `result`.

Values (e.g., `result`) are returned with `return result`.

testing a function

If we store the function definition for `fun`
in the file `ourfirstmodule.py`, we can do

```
>>> from ourfirstmodule import fun
>>> fun(1,2,3)
67.18943930405324
```

We *import* the function `fun` into an interactive Python session.

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

help(ourfirstmodule)

```
Help on module ourfirstmodule:

NAME
    ourfirstmodule

DESCRIPTION
    A module is a collection of functions.
    This module exports two functions: fun and main.
    By the last line of this file, we can run this
    script at the command prompt $ as
    $ python ourfirstmodule.py.
    Alternatively, we can import fun and/or main
    in an interactive Python session.
```

What follows under `DESCRIPTION` is the documentation string of the module, stored in the variable `__doc__`.

help continued

```
FUNCTIONS
    fun(xarg, yarg, zarg)
        Returns the value of the expression
          xarg**2*cos(yarg) + 4*exp(zarg)*sin(xarg)
        for numerical values of the formal arguments
        xarg, yarg, and zarg.

    main()
        Prompts the user for three values
        for the variables x, y, z and
        prints x**2*cos(y) + 4*exp(z)*sin(x).
```

the main function

```
def main():
    """
    Prompts the user for three values
    for the variables x, y, z and
    prints x**2*cos(y) + 4*exp(z)*sin(x).
    """
    print('v = x**2*cos(y) + 4*exp(z)*sin(x)')
    xval = float(input('give x : '))
    yval = float(input('give y : '))
    zval = float(input('give z : '))
    fval = fun(xval, yval, zval)
    print('v = ', fval)
```

At the time of the function call `fun(xval, yval, zval)`, the values of `xval`, `yval`, `zval` are assigned to the formal arguments `xarg`, `yarg`, and `zarg` of the function definition. The variables `xval`, `yval`, `zval` are the actual arguments of the function.

running `main()`

We can also import the function `main()`.
In order to run `main()` as a program
at the command prompt `$`

```
$ python ourfirstmodule.py
```

the last line in `ourfirstmodule.py` is

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

the module search path

Where does Python find its modules?

The module `sys` exports the *module search path*
as the dynamic object `path`.

```
>>> import sys
>>> sys.path
['', '...', ...]
```

`sys.path[0]` is the script directory, or `''`.

The order of the directories in the list returned by `sys.path`
determines the order in which Python searches for modules to import.

my math module

```
def test():
    print("This is my math module!")

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

If we save the code above as `math.py` in the current directory where we call the python interpreter, then:

```
>>> import math
>>> math.test()
This is my math module!
>> from math import cos
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name cos
```

doing an “un-import”

What if we want to import the right `math` module?

```
>>> import math
>>> math.test()
This is my math module!
>>> import sys
>>> sys.modules['math']
<module 'math' from 'math.py'>
>>> del sys.modules['math']
```

`modules` is a dictionary of loaded modules, removing `path[0]`:

```
>>> sys.path = sys.path[1:]
>>> from math import cos
>>> cos(3)
-0.9899924966004454
```

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

extending Python with your own package

A package is

- a directory containing an `__init__.py` file; and
- also modules or other packages.

The code in a Python package can be

- a pure Python module, defined in a single `.py` file; or
- a shared object `.so` file with compiled C code.

The directory `site-packages` contains installed packages, such as numpy, scipy, sympy, etc.

The Hitchiker's Guide to Packaging 1.0 documentation is at
https://the-hitchhikers-guide-to-packaging.readthedocs.io/en/latest/

preparing a package for distribution

Distutils is

- a standard and basic package in the Python standard library,
- used for creating distributions, imported in the `setup.py` file.

Following the Hitchhiker's Guide to Packaging:

1. Define the directory structure:
   - Make a directory with `LICENSE.txt`, `MANIFEST.in`, `README.txt`, `setup.py`.
     The name for this directory could be the package name, followed by the version number, e.g.: `ourfirstpackage-0.0.1`.
     In this directory is a subdirectory with the package name.
   - The directory with the package name contains the `__init__.py` and the module with the code.
2. `python setup.py sdist` makes a file `MANIFEST` and a directory `dist` with a gzipped tar file.
3. As superuser we install with `python setup.py install`.

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

making `ourfirstpackage`

We use `ourfirstmodule.py`.

1. New directory: `mkdir ourfirstpackage-0.0.1`.
2. In `ourfirstpackage-0.0.1`, do `mkdir ourfirstpackage` and copy `ourfirstmodule.py` into `ourfirstpackage`.
3. As `LICENSE.txt`, we copy the text of GNU GPL version 3.
4. The content of `README.txt` (start documenting in *reST*):

```
============================
Welcome to ourfirstpackage!
============================

It contains ourfirstmodule of lecture 17 of MCS 507.

ourfirstmodule
--------------

This module exports x**2*cos(y) + 4*exp(z)*sin(x).
```

the file `MANIFEST.in` and `__init__.py`

- The contents of the file `MANIFEST.in`:

  ```
  include *.txt
  ```

  So all .txt files are included in the distribution.

- The contents of the file `__init__.py`:

  ```
  """
  ourfirstpackage
  ===============

  Exports the function x**2*cos(y) + 4*exp(z)*sin(x)
  in the module ourfirstmodule.
  """

  # Definition of the version number
  __version__ = '0.0.1'
  ```

the file `setup.py`

```
from distutils.core import setup
from distutils.sysconfig import get_python_lib

setup(
    name = 'ourfirstpackage' ,
    author = 'MCS 507' ,
    author_email = 'janv@uic.edu' ,
    description = 'our first package' ,
    url = 'http://www.math.uic.edu/~jan/mcs507' ,
    version = '0.0.1' ,
    packages = ['ourfirstpackage'] ,
    py_modules = ['ourfirstpackage/ourfirstmodule'] ,
    license = 'GNU GENERAL PUBLIC LICENSE version 3' ,
    platforms = ['linux2'] ,
    long_description=open('README.txt').read()
)
```

preparing the distribution

```
$ python setup.py sdist
running sdist
running check
reading manifest template 'MANIFEST.in'
writing manifest file 'MANIFEST'
creating ourfirstpackage-0.0.1
creating ourfirstpackage-0.0.1/ourfirstpackage
making hard links in ourfirstpackage-0.0.1...
hard linking LICENSE.txt -> ourfirstpackage-0.0.1
hard linking README.txt -> ourfirstpackage-0.0.1
hard linking setup.py -> ourfirstpackage-0.0.1
hard linking ourfirstpackage/__init__.py -> ourfirstpackage
hard linking ourfirstpackage/ourfirstmodule.py -> ourfirstp
creating dist
Creating tar archive
removing 'ourfirstpackage-0.0.1' (and everything under it)
$ ls dist
ourfirstpackage-0.0.1.tar.gz
$
```

installing the package, as superuser

```
$ python setup.py install
running install
running build
running build_py
creating build
creating build/lib
creating build/lib/ourfirstpackage
copying ourfirstpackage/ourfirstmodule.py -> build/lib/\
ourfirstpackage
copying ourfirstpackage/__init__.py -> build/lib/\
ourfirstpackage
running install_lib
running install_egg_info
... etc ...
$
```

using ourfirstpackage

```
$ python
>>> import ourfirstpackage
>>> help(ourfirstpackage)

>>> from ourfirstpackage import ourfirstmodule
>>> help(ourfirstmodule)

>>> from ourfirstpackage.ourfirstmodule import main
>>> main()
v = x**2*cos(y) + 4*exp(z)*sin(x)
give x : 1
give y : 2
give z : 3
v =  67.1894393041
>>>
```

help on ourfirstpackage

```
Help on package ourfirstpackage:

NAME
    ourfirstpackage

DESCRIPTION
    ourfirstpackage
    ===============

    Exports the function x**2*cos(y) + 4*exp(z)*sin(x)
    in the module ourfirstmodule.

PACKAGE CONTENTS
    ourfirstmodule

DATA
    __version__ = '0.0.1'

VERSION
    0.0.1
```

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

History of Open Source

In the 1960s, software came free with hardware.
Users were programmers and software developers.

In the mid 1970s, software became a commercial product.
Personal Computing meant that users did not program.

Unix, developed at Bell Labs and University of Berkeley (BSD)
became a commercial product of AT&T in 1984.

GNU (not Unix) project started at MIT in 1984 by Richard Stallman.
The Free Software Foundation published a first version of the General Public License (GPL) in 1988.

In 1991, a first version of what later became Linux is released.
Linux = Linus+Unix, started by Linus Torvalds.

MIT started the OpenCourseWare project in 2000.

Intellectual Property Rights

and software

Intellectual property rights (IPR) grant authors the possibility to control copying, modifying, and distributing products.

Four tools to protect IPR:

trade secret	confidential information used to compete
trademark	is word, name, symbol used to distinguish
copyright	grants authors exclusive rights to copy, sell, license, distribute, modify, translate, etc.
patent	gives creators exclusive rights to manufacture and sell invented goods.

A commercial program is typically delivered by giving only its object code, in contrast to open source.
Licenses may explicity forbid reverse engineering of code.

General Public License (GPL)

Motivation: needs and benefits of open source.

1. Mission critical and scientific applications place a very strong emphasis on correctness.
2. *Given enough eyeballs, all bugs are shallow*.

Note: free as in free speech, not as in free beer.

Software is also a service.

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

copy*left* versus copy*right*

Copy*left* as the reverse of copy*right*:

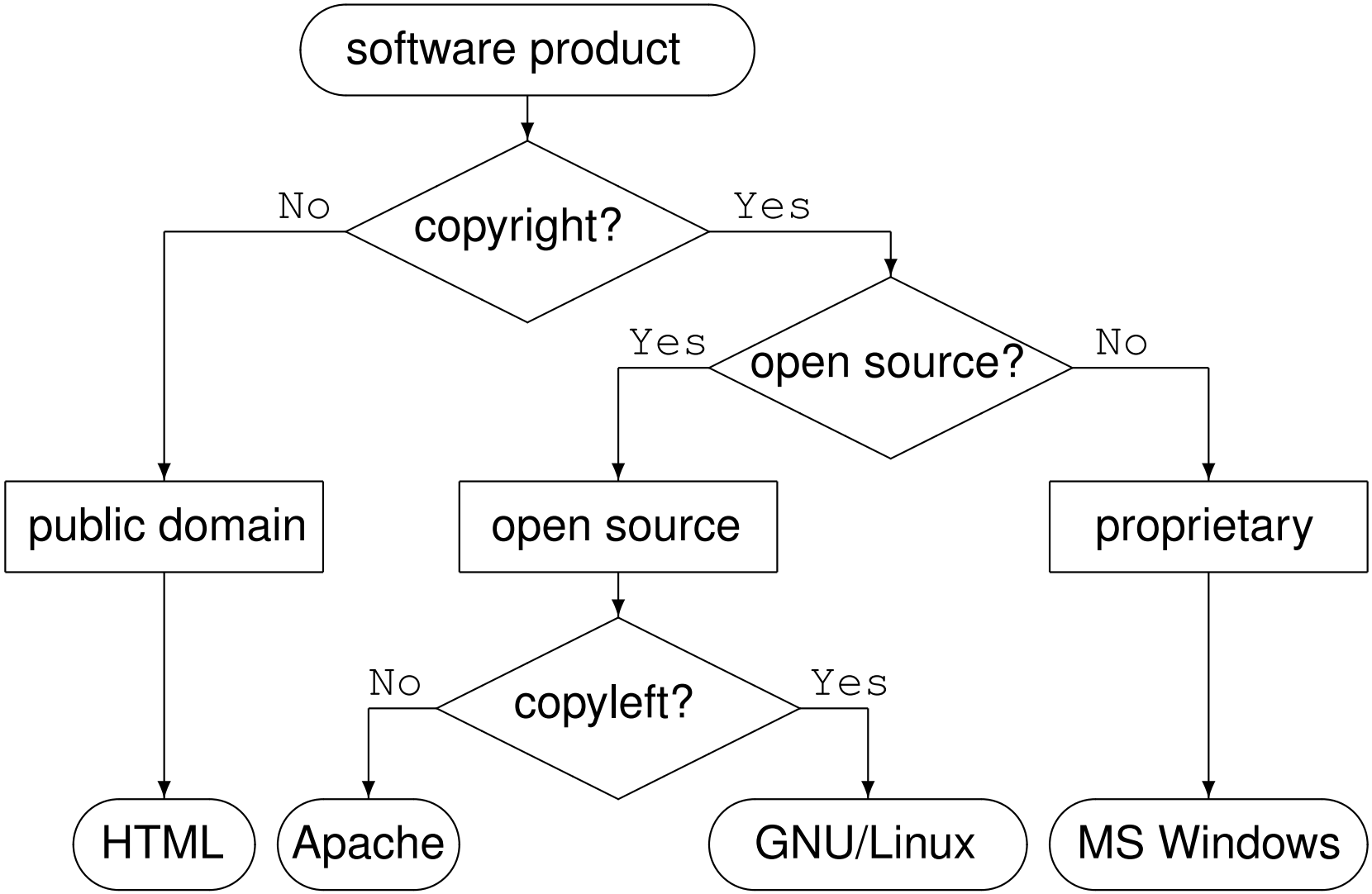
1. first state that the code is copyrighted
2. give everyone all rights on the code
3. *only if the distribution terms are unchanged*

The code and the freedoms become legally inseparable.

The license imposes a ‘share alike’ restriction.

Many open source licenses exist, some of the most popular ones: GNU GPL, GNU Library or Lesser GPL, Mozilla Public License, and Berkeley Software Distribution.

copyright applied to software



M. Muffatto: *Open Source. A Multidisciplinary Approach*, 2006.

some reading suggestions

- T. Basaglia, Z. W. Bell, A. Larkin, and M. G. Pia.
  Writing Software or Writing Scientific Articles?
  IEEE Transactions on Nuclear Science, 55(2):671–678, 2008.
- Reynold Xin, Wes McKinney, Alan Gates, and Chris McCubbin.
  It Takes a Community: The Open-Source Challenge.
  ACM Queue, pages 115–136, September-October 2021.

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

four parts of the manual

A good software manual has four parts:

1. installation guide
2. getting started
3. tutorial
4. reference manual

Packaging Python code and Sphinx

1. Functions and Modules
   - writing a function definition
   - defining and using modules
2. Python packages
   - extending Python with your own package
   - making `ourfirstpackage`
3. Open Source Software
   - licensing software
   - copyright and copyleft
4. Documenting Software with Sphinx
   - the manual
   - Sphinx generates documentation

Sphinx generates documentation

What is Sphinx? From `http://sphinx-doc.org`:

- Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license.
- It was originally created for the new Python documentation, and it has excellent facilities for the documentation of Python projects, but other languages are supported as well.
- Sphinx uses reStructuredText as its markup language, and many of its strengths come from the power and straightforwardness of reStructuredText and its parsing and translating suite, the Docutils.

The *same* documentation is generated in several formats, including latex, pdf, html.

Why Sphinx? Used for the python language, numpy, sympy, matplotlib.

step-by-step documentation generation

Given Python code with documentation strings,
generate documentation with Sphinx.

There are a few steps to take:

1. Make a new directory `doc` and do `"cd doc"`.
2. Run `sphinx-quickstart` in the `doc` directory.
   Following default options, `doc` contains a `"Makefile"`
   and two directories: `"build"` and `"source"`.
3. Edit `conf.py` in `source` as follows:

   ```
   sys.path.insert(0, os.path.abspath('<path>'))
   ```

   where `<path>` is the directory where our Python code is.
4. Edit `index.rst` with `".. automodule::"` lines.
5. Type `"make latexpdf"` or `"make html"` to generate.

Summary and Exercises

We do not release source code without documentation.

Sphinx generates documention automatically.

Use Sphinx in your computer projects!

Exercises:

- Take the Python scripts you used in your first computer project and make a Python package.
- Document your Python package with restructured text and use Sphinx to generate documentation in html and latex-pdf formats.