Scientific Computing Using Python

Development practices and tools

14th of June, 2023

Thomas Arildsen
tari@its.aau.dk

CLAAUDIA
Aalborg University

Introduction

Documentation via Sphinx

"Nice Code"

Code Complexity

PEP8 – Style Guide for Python Code

Static Code Checkers

Documentation via Sphinx

Key packages

Sphinx – documentation

Sphinx – what it offers ...

- Full Python documentation system – combination of manual and auto-generated information.
- Various output formats – **HTML**, **LaTeX**, **PDF**.
- Automatically generated index, content and search facilities.
- Automatic code handling – insert full code parts or only specific code parts.
- Based on **reStructuredText** – fairly easy conversion to HTML and LaTeX.

Key packages

Sphinx – documentation

Get started

- Decide on the root directory – recommended folder structure:
  `parentfolder/doc` ← for documentation
  `parentfolder/packagename`
- In a terminal window start with (in the folder `parentfolder/`):
  `$ sphinx-quickstart -ext-autodoc -ext-mathjax doc`
  `$ sphinx-apidoc -o doc/ sorts`
- Edit `conf.py`, uncomment the lines concerning `sys.path` and add:
  `sys.path.append(os.path.abspath('..'))`
- Edit `index.rst` to include '`modules`' below the '`toctree`' directive.
- When files have been updated run e.g. **make html**.

Key packages

Sphinx – documentation

- Directory structure:

```
drwxrwxr-x 5 tlj tlj 4096 May 14 12:30 .
drwxrwxr-x 3 tlj tlj 4096 May 14 12:30 ..
drwxrwxr-x 4 tlj tlj 4096 May 13 23:18 _build
-rw-rw-r-- 1 tlj tlj 8326 May 14 11:03 conf.py
-rw-rw-r-- 1 tlj tlj  204 May 14 12:30 csorting.rst
-rw-rw-r-- 1 tlj tlj  456 May 14 11:19 index.rst
-rw-rw-r-- 1 tlj tlj  409 May 14 11:11 indexsort.rst
-rw-rw-r-- 1 tlj tlj 6699 May 13 23:12 make.bat
-rw-rw-r-- 1 tlj tlj 6758 May 13 23:12 Makefile
drwxrwxr-x 2 tlj tlj 4096 May 13 23:12 _static
drwxrwxr-x 2 tlj tlj 4096 May 13 23:12 _templates
```

- Critical files:
  - **index.rst** – content and outline of include files etc.
  - **conf.py** – configuration file.

Key packages

Sphinx – documentation

- **index.rst** content:

```
.. Sorts documentation master file, created by
   sphinx-quickstart on Tue Jun 16 19:15:13 2020.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Welcome to Sorts's documentation!
=================================

.. toctree::
   :maxdepth: 2
   :caption: Contents:

   modules

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Key packages

Sphinx – documentation

- Example of **sorts.rst** content:

```
sorts.indexsort module
----------------------

.. automodule:: sorts.indexsort
   :members:
   :undoc-members:
   :show-inheritance:
```

- Characteristics:
  - Module name.
  - Details on function automatically extracted from docstring.
  - Interpreted using reStructuredText (**.rst** files).
  - Possible to see the code via link in HTML. Selected parts of the code can be extracted by identifiers in the code, if you wish.

Key packages

Sphinx – documentation

- Docstring uses reStrucutedText heavily, which is not always very readable in the "raw" docstring.
- Trade-off:
  - Follow raw text style ala Google style guide [Pat+] and accept less than beautiful formatting
  - Maintain two sets of documentation.
  - Minimum "restructuring" that is readable both in raw and via Sphinx. *See example on next slide.*

Key packages

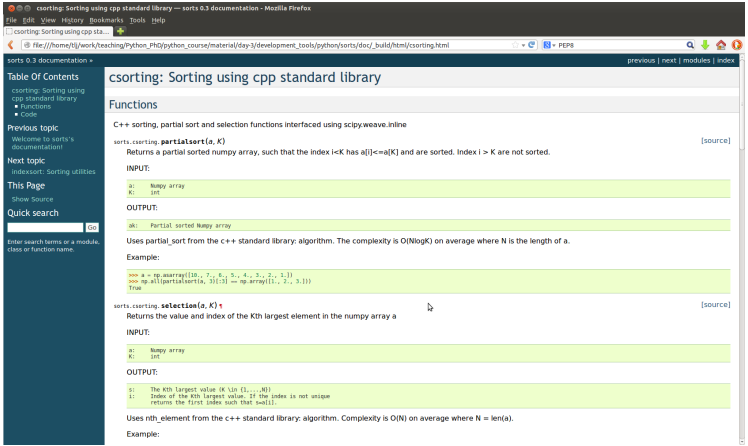
Sphinx – documentation


```python
def selection(a, K):
    """
    Returns the value and index of the Kth largest element in the array a

    INPUT::

     a:     Numpy array
     K:     int

    OUTPUT::

     s:     The Kth largest value (K in {1,...,N})
     i:     Index of the Kth largest value. If the index is not unique
            returns the first index such that s=a[i].

    Uses nth_element from the c++ standard library: algorithm. Complexity
    is O(N) on average where N = len(a).

    Example:

    >>> a = np.asarray([10., 7., 6., 5., 4., 3., 2., 1.])
    >>> selection(a, 1)
    (10.0, 0)
    >>> selection(a, 4)
    (5.0, 3)

    """
```

Key packages

Sphinx – documentation

- Result:



“Nice Code”

"Nice Code"

It is important to write code that it understandable.
Quoting from [Wil+14]:

1. *Write programs for people, not computers.*
   - *A program should not require its readers to hold more than a handful of facts in memory at once.*
   - *Make names consistent, distinctive, and meaningful.*
   - *Make code style and formatting consistent.*

See also, e.g. [San+13; SM14].

Code Complexity

Code Complexity

Cyclomatic complexity

- Complexity here means: the more complex, the more difficult to read and understand.
- We can use cyclomatic complexity as an indicator of how readable our code is.

Code Complexity

Cyclomatic complexity

- Based on the control flow graph of a program.

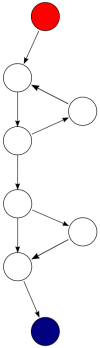


Figure: Control flow graph. Source: Wikipedia.

Code Complexity

Cyclomatic complexity

- Quoting Wikipedia:

  The complexity M is then defined as

$$M = E - N + 2P,$$

where

E *the number of edges of the graph.*
N *the number of nodes of the graph.*
P *the number of connected components.*

Code Complexity

Evaluating cyclomatic complexity

We can use the package `radon` to calculate cyclomatic complexity for our code: `https://pypi.python.org/pypi/radon`.

- Easily installed via `conda` (from command line: `conda install radon`) or `pip`.
- Command-line utility to check code.
- Run as: `radon cc script.py`

PEP8 – Style Guide for Python Code

PEP8

- "Readability counts"[1]
- Consistency (but remain consistent with the rest of the code).
- Programs that can check your code against the PEP8 style guide.

What is PEP? *Python enhancement proposal*.

[1] https://www.python.org/dev/peps/pep-0020

PEP8

A few points comes a long way

- Indent using four spaces.
- Code no more than 79 characters, docstring 72.
- docstring on the following line of e.g. a function definition.
- Lower-case variable names with underscore separating words.
- Make a single space around =, == etc.
- Inline comments as `<code>  # comment`.

PEP8

Checking compliance

The program `pycodestyle` can be used to check your code for PEP8 compliance: `https://pypi.python.org/pypi/pep8`.

- Included in Anaconda.
- Command-line utility to check code.
- Run as: `pycodestyle script.py`

Static Code Checkers

Static Code Checkers

Static code checkers are programs that check our code for consistency, bad coding style/practice etc. without actually executing the code.

- Various alternatives exist.
- `pylint` seems like the most up-to-date for Python code.
- Another good option is `flake8`

Static Code Checkers

pylint

The program `pylint` can be used to check your code for various style issues, bad coding practice etc.: `http://www.pylint.org/`.

- Included in Anaconda.
- Command-line utility to check code.
- Run as: `pylint script.py`

References I

[Pat+] Amit Patel et al. *Google Python Style Guide*. Version 2.59. URL: https://google.github.io/styleguide/pyguide.html.

[San+13] Geir Kjetil Sandve et al. "Ten Simple Rules for Reproducible Computational Research". In: *PLoS Comput Biol* 9.10 (Oct. 2013), e1003285. DOI: 10.1371/journal.pcbi.1003285.

[SM14] Victoria Stodden and Sheila Miguez. "Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research". In: *Journal of Open Research Software* 2.1 (July 2014), e21. DOI: 10.5334/jors.ay.

[Wil+14] Greg Wilson et al. "Best Practices for Scientific Computing". In: *PLoS Biol* 12.1 (Jan. 2014), e1001745. DOI: 10.1371/journal.pbio.1001745.