Documenter un projet Python

Présentation des outils les plus utilisées

Auteur: Jérémie DECOCK

Contact: jd.jdhp@gmail.com

Version: 0.1

Date: 29/05/2016

Sommaire

Avis

Veuillez noter que ce document est en cours de rédaction. Dans son état actuel, il n'est pas destiné à être lu par d'autres personnes que ses auteurs.

1 Introduction

Si vous vous demendez en quoi c'est important de documenter un projet, je vous conseille le lien suivant (en anglais): http://write-the-docs.readthedocs.io/guide/writing/beginners-guide-to-docs/#why-write-docs.

2 Docstrings

2.1 Brève description

Équivalant de javadoc (Java) et de doxygen (C++, ...)

2.2 Philosophie des docstrings

- Documentation au plus près du code

2.3 Bonnes pratiques

Une docstring doit être associée à chaque

- module
- function
- classe
- methode

d'un projet Python.

Pour plus d'information, voir https://www.python.org/dev/peps/pep-0257/#what-is-a-docstring

2.4 Exemple

```python
# -*- coding: utf-8 -*-

"""
This module does blah blah.

Put here the description of the module.
"""

def main():
    """
    This function does blah blah.

    Put here the description of the function.
    """

    print("Hello!")
```

2.5 Documents de références

Guides de référence pour la rédaction des docstrings en Python (*PEPs actives*):

- PEP8 "Python's good practices"
- PEP257 "Docstring Conventions"
- PEP287 "reStructuredText Docstring Format"

À titre d'information, les *PEPs* suivantes traitent aussi du sujet mais ont été rejetées ou sont dépréciées:

- PEP216 "Docstring Format" (remplacée par la PEP287)
- PEP224 "Attribute Docstrings" (rejetée)
- PEP256 "Docstring Processing System Framework" (rejetée)
- PEP258 "Docutils Design Specification" (rejetée)

2.6 Quelques mots à propos des *doctests*

Les docstrings python peuvent contenir des *doctests*.

Ces *doctests* permettent d'intégrés des *tests unitaires* à la documentation (i.e. dans les docstrings).

Ils peuvent être utilisés comme alternative ou comme complément au framework *unittest*.

Pour plus d'information, consulter https://docs.python.org/3/library/doctest.html.

3 reStructuredText

3.1 Présentation

ReStructuredText est un langage de balisage léger utilisé pour ajouter la possibilité de mise en forme avancée aux docstrings (italique, gras, listes, tableaux, images, ...). Depuis la PEP287 ("reStructuredText Docstring Format") c'est devenu un standard pour la rédaction des docstrings Python.

Ce langage est aussi utilisé pour documenter plus largement les projets Python (c'est à dire au delà de l'API). On peut l'utiser en dehors des docstrings et écrire des documents complets en reStructuredText (fichiers .rst). Par exemple, il est utilisé pour écrire le site web des projets Python, Matplotlib, Scikit-Learn, etc.

ReStructuredText est l'équivalant d'un autre langage de balisage léger très populaire: Markdown.

Pour plus d'information, voir http://docutils.sourceforge.net/rst.html.

4 Les outils

Cette partie présente les principaux outils utilisés pour générer la documentation d'un projet à partir des docstrings dans un format plus agréable à lire et plus facile à partager pour les utilisateurs (HTML, PDF, ...).

4.1 Docutils

Les *docutils* sont des outils distribués avec Python pour convertir les fichiers reStructuredText en HTML, LaTeX, etc.

Ce sont les outils de référence pour convertir les documents reStructuredText mais il en existe d'autres développés par des tiers : Pandoc, rst2pdf, etc.

4.2 Pandoc

Pandoc est un logiciel capable de traduire un grand nombre de formats de documents.

Il permet entre autre de convertir des fichiers Markdown, HTML, Latex ou DocBook en documents reStructuredText et inversement.

Pandoc peut être utilisé comme alternative aux docutils.

Pour plus d'information, voir http://pandoc.org/.

4.3 PyDoc

Remplacé depuis longtemps par les docutils...

https://docs.python.org/3/library/pydoc.html

4.4 Epydoc

http://epydoc.sourceforge.net/

Epydoc est un équivalant de javadoc et doxygen pour Python. Il ne gère que la documentation de l'API contrairement a sphinx.

Epydoc semble mort depuis plusieurs années, la dernière version (et la dernière news) date de janvier 2008... La dernière version d'epydoc ne serait plus compatible avec les dernières versions de docutils... Je ne sais pas si il marche avec Python 3.

La communauté Python semble avoir largement abandonné epydoc au profit de sphinx: - http://stackoverflow.com/questions/4126421/how-to-document-python-code-epydoc-doxygen-sphinx - http://stackoverflow.com/questions/5579198/should-we-use-epydoc-sphinx-or-something-else-for-documenting-python-apis

Tutoriel en français: http://deusyss.developpez.com/tutoriels/Python/Epydoc/

4.5 Sphinx

http://sphinx-doc.org/

> "The focus is on hand-written documentation, rather than auto-generated API docs. Though there is support for that kind of docs as well (which is intended to be freely mixed with hand-written content), if you need pure API docs have a look at *Epydoc*, which also understands reST." (http://sphinx-doc.org/intro.html#prerequisites)

4.5.1 Premiers pas

Générer la doc:

```
mkdir docs
sphinx-quickstart
```

4.5.2 Autodoc

- http://sphinx-doc.org/tutorial.html#autodoc
- http://sphinx-doc.org/ext/autodoc.html

Lorsqu'on utilise:

```
.. automodule:: foo.bar
    :members:
```

seuls les membres définis dans __all__ sont importés.

Les docstring des constructeurs de classes __init__() sont ignorés par défaut. En fait, il n'est pas nécessaire/recommandé de mettre une docstring aux constructeur (pylint ne râle pas), la description des arguments du constructeur doit à priori être faite dans la docstring de la classe.

On peut néanmoins obliger sphinx à générer la documentation des constructeurs: http://stackoverflow.com/questions/5599254/how-to-use-sphinxs-autodoc-to-document-a-classs-init-self-method

Les PEPs ne définissent pas vraiment le contenu des docstring, seulement le langage de markup et l'intégration des docstrings dans le code. Autrement dit, il n'y a pas de notation standard pour decrire les parametre, la valeur de retour, etc (contrairement à javadoc par exemple).

Le mieux est alors de s'inspirer de ce qui ce fait de mieux...

Styles de docstring:

4.5.2.1 Pillow

(ex: https://github.com/python-pillow/Pillow/blob/master/PIL/Image.py) et PIP (https://github.com/pypa/pip/blob/develop/pip/index.py), simple et élégant

```
def getmodebandnames(mode):
    """
    Gets a list of individual band names.  Given a mode, this function returns
    a tuple containing the names of individual bands (use
    :py:method:`~PIL.Image.getmodetype` to get the mode used to store each
    individual band.

    :param mode: Input mode.
    :returns: A tuple containing band names.  The length of the tuple
        gives the number of bands in an image of the given mode.
    :exception KeyError: If the input mode was not a standard mode.
    """
    return ImageMode.getmode(mode).bands
```

4.5.2.2 Matplotlib

(ex: https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/animation.py)

```
def __init__(self, fps=5, codec=None, bitrate=None, extra_args=None, metadata=None):
    '''
    Construct a new MovieWriter object.

    fps: int
        Framerate for movie.
    codec: string or None, optional
        The codec to use. If None (the default) the setting in the
        rcParam `animation.codec` is used.
    bitrate: int or None, optional
        The bitrate for the saved movie file, which is one way to control
        the output file size and quality. The default value is None,
        which uses the value stored in the rcParam `animation.bitrate`.
        A value of -1 implies that the bitrate should be determined
        automatically by the underlying utility.
    extra_args: list of strings or None
        A list of extra string arguments to be passed to the underlying
        movie utiltiy. The default is None, which passes the additional
        argurments in the 'animation.extra_args' rcParam.
    metadata: dict of string:string or None
        A dictionary of keys and values for metadata to include in the
        output file. Some keys that may be of use include:
        title, artist, genre, subject, copyright, srcform, comment.
    '''
```

4.5.2.3 Mayavi

(ex: https://github.com/enthought/mayavi/blob/master/mayavi/core/engine.py)

```
def add_scene(self, scene, name=None):
    """Add given `scene` (a `pyface.tvtk.scene.Scene` instance) to
    the mayavi engine so that mayavi can manage the scene.  This
    is used when the user creates a scene.  Note that for the
    `EnvisageEngine` this is automatically taken care of when you
    create a new scene using the TVTK scene plugin.

    Parameters:
    -----------

     scene - `pyface.tvtk.scene.Scene`

       The scene that needs to be managed from mayavi.

     name - `str`

       The name assigned to the scene.  It tries to determine the
       name of the scene from the passed scene instance.  If this
       is not possible it defaults to 'Mayavi Scene'.
    """
```

4.5.2.4 Scipy

(ex: https://github.com/scipy/scipy/blob/master/scipy/linalg/decomp_cholesky.py)

```
def cholesky(a, lower=False, overwrite_a=False, check_finite=True):
    """
    Compute the Cholesky decomposition of a matrix.

    Returns the Cholesky decomposition, :math:`A = L L^*` or
    :math:`A = U^* U` of a Hermitian positive-definite matrix A.

    Parameters
    ----------
    a : (M, M) array_like
        Matrix to be decomposed
    lower : bool, optional
        Whether to compute the upper or lower triangular Cholesky
        factorization.  Default is upper-triangular.
    overwrite_a : bool, optional
        Whether to overwrite data in `a` (may improve performance).
    check_finite : bool, optional
        Whether to check that the input matrix contains only finite numbers.
        Disabling may give a performance gain, but may result in problems
        (crashes, non-termination) if the inputs do contain infinities or NaNs.

    Returns
    -------
    c : (M, M) ndarray
        Upper- or lower-triangular Cholesky factor of `a`.

    Raises
    ------
    LinAlgError : if decomposition fails.

    Examples
    --------
    >>> from scipy import array, linalg, dot
    >>> a = array([[1,-2j],[2j,5]])
    >>> L = linalg.cholesky(a, lower=True)
    >>> L
    array([[ 1.+0.j,  0.+0.j],
           [ 0.+2.j,  1.+0.j]])
    >>> dot(L, L.T.conj())
    array([[ 1.+0.j,  0.-2.j],
           [ 0.+2.j,  5.+0.j]])

    """
```

4.5.2.5 Numpy

http://sphinxcontrib-napoleon.readthedocs.org/en/latest/example_numpy.html

```
def module_level_function(param1, param2=None, *args, **kwargs):
"""This is an example of a module level function.

Function parameters should be documented in the ``Parameters`` section.
The name of each parameter is required. The type and description of each
parameter is optional, but should be included if not obvious.
```

```
Parameter types -- if given -- should be specified according to
`PEP 484`_, though `PEP 484`_ conformance isn't required or enforced.

If \*args or \*\*kwargs are accepted,
they should be listed as ``*args`` and ``**kwargs``.

The format for a parameter is::

    name : type
        description

        The description may span multiple lines. Following lines
        should be indented to match the first line of the description.
        The ": type" is optional.

        Multiple paragraphs are supported in parameter
        descriptions.

Parameters
----------
param1 : int
    The first parameter.
param2 : Optional[str]
    The second parameter.
*args
    Variable length argument list.
**kwargs
    Arbitrary keyword arguments.

Returns
-------
bool
    True if successful, False otherwise.

    The return type is not optional. The ``Returns`` section may span
    multiple lines and paragraphs. Following lines should be indented to
    match the first line of the description.

    The ``Returns`` section supports any reStructuredText formatting,
    including literal blocks::

        {
            'param1': param1,
            'param2': param2
        }

Raises
------
AttributeError
    The ``Raises`` section is a list of all exceptions
    that are relevant to the interface.
ValueError
    If `param2` is equal to `param1`.


.. _PEP 484:
```

```
   https://www.python.org/dev/peps/pep-0484/

"""
if param1 == param2:
    raise ValueError('param1 may not be equal to param2')
return True
```

(ex: https://github.com/numpy/numpy/blob/master/numpy/core/numeric.py)

```
def full_like(a, fill_value, dtype=None, order='K', subok=True):
    """
    Return a full array with the same shape and type as a given array.

    Parameters
    ----------
    a : array_like
        The shape and data-type of `a` define these same attributes of
        the returned array.
    fill_value : scalar
        Fill value.
    dtype : data-type, optional
        Overrides the data type of the result.
    order : {'C', 'F', 'A', or 'K'}, optional
        Overrides the memory layout of the result. 'C' means C-order,
        'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,
        'C' otherwise. 'K' means match the layout of `a` as closely
        as possible.
    subok : bool, optional.
        If True, then the newly created array will use the sub-class
        type of 'a', otherwise it will be a base-class array. Defaults
        to True.

    Returns
    -------
    out : ndarray
        Array of `fill_value` with the same shape and type as `a`.

    See Also
    --------
    zeros_like : Return an array of zeros with shape and type of input.
    ones_like : Return an array of ones with shape and type of input.
    empty_like : Return an empty array with shape and type of input.
    zeros : Return a new array setting values to zero.
    ones : Return a new array setting values to one.
    empty : Return a new uninitialized array.
    full : Fill a new array.

    Examples
    --------
    >>> x = np.arange(6, dtype=np.int)
    >>> np.full_like(x, 1)
    array([1, 1, 1, 1, 1, 1])
    >>> np.full_like(x, 0.1)
    array([0, 0, 0, 0, 0, 0])
    >>> np.full_like(x, 0.1, dtype=np.double)
```

```
    array([ 0.1,  0.1,  0.1,  0.1,  0.1,  0.1])
    >>> np.full_like(x, np.nan, dtype=np.double)
    array([ nan,  nan,  nan,  nan,  nan,  nan])

    >>> y = np.arange(6, dtype=np.double)
    >>> np.full_like(y, 0.1)
    array([ 0.1,  0.1,  0.1,  0.1,  0.1,  0.1])

    """
    res = empty_like(a, dtype=dtype, order=order, subok=subok)
    multiarray.copyto(res, fill_value, casting='unsafe')
    return res
```

Il semble y avoir 2 principaux styles éprouvés:

- la première méthode (Pillow, PIP) semble être le sytle officiel recommandé par sphinx:
    - cf. http://stackoverflow.com/questions/5334531/python-documentation-standard-for-docstring
    - cf. http://sphinx-doc.org/domains.html#info-field-lists
- un style alternatif semble rencontrer un certain succès aussi: celui de Numpy et Google (cf. section "Napoleon" ci-dessous).

4.5.3 Napoleon is a Sphinx extension

Numpy et Google (entre autre) utilisent *Napoleon*, un style éprouvé différent de celui recommandé par sphinx:

- http://sphinxcontrib-napoleon.readthedocs.org/
- http://sphinx-doc.org/ext/napoleon.html#module-sphinx.ext.napoleon
- http://sphinxcontrib-napoleon.readthedocs.org/en/latest/example_numpy.html
- http://sphinxcontrib-napoleon.readthedocs.org/en/latest/example_google.html#example-google

> Napoleon is a pre-processor that parses NumPy and Google style docstrings and converts them to reStructuredText before Sphinx attempts to parse them. This happens in an intermediate step while Sphinx is processing the documentation, so it doesn't modify any of the docstrings in your actual source code files. (http://sphinxcontrib-napoleon.readthedocs.org)

4.5.4 Thèmes

http://docs.writethedocs.org/tools/sphinx-themes/

Builtin themes (cf. http://sphinx-doc.org/theming.html):

- basic
- alabaster
- sphinx_rtd_theme
- classic
- sphinxdoc
- scrolls
- agogo
- nature
- pyramid
- haiku

- traditional
- epub
- bizstyle

3 themes semblent avoir la cote:

- Read the Docs Theme https://github.com/snide/sphinx_rtd_theme
- Alabaster https://github.com/bitprophet/alabaster
- Sphinx Bootstrap Theme https://github.com/ryan-roemer/sphinx-bootstrap-theme

Thèmes disponibles sur Debian 8 (nom des paquets):

- python3-sphinx-rtd-theme
- python-guzzle-sphinx-theme
- doctrine-sphinx-theme
- python-oslosphinx

4.5.5 Extensions disponibles sur Debian 8

Plusieurs paquets fournissent des extensions à Sphinx sur Debian 8, voici leur nom:

- python3-sphinxcontrib.actdiag
- python3-sphinxcontrib.blockdiag
- python3-sphinxcontrib.nwdiag
- python3-sphinxcontrib.programoutput
- python3-sphinxcontrib.seqdiag
- python3-sphinxcontrib.youtube

4.5.6 Odt2sphinx

https://pypi.python.org/pypi/odt2sphinx/ (http://sphinx-doc.org/intro.html#conversion-from-other-systems)

5 Les services

5.1 Readthedocs.org

5.1.1 À quoi ça sert ?

Readthedocs est un service souvent utilisé par les dévelopeurs de projets Python pour héberger la documentation de leurs projets au format HTML (sur le site readthedocs.org).

Le principal intéret de ce service est qu'il regénére et publie automatiquement la documentation d'un projet chaque fois que son code source change (i.e. à chaque commit Git).

Pour plus d'information, voir http://write-the-docs.readthedocs.io/guide/.

6 Inspiration

Projets ayants soigné la rédaction du ficher README:

- Scikit-learn (https://github.com/scikit-learn/scikit-learn)
  - 3 webbadges:
    - https://travis-ci.org/scikit-learn/scikit-learn

        - https://ci.appveyor.com/project/sklearn-ci/scikit-learn/history
        - https://coveralls.io/r/scikit-learn/scikit-learn
    - section "Important links"
    - section "Dependencies"
- IPython (https://github.com/ipython/ipython)
    - 4 webbadges:
        - https://coveralls.io/r/ipython/ipython?branch=master
        - https://pypi.python.org/pypi/ipython
        - https://pypi.python.org/pypi/ipython
        - https://travis-ci.org/ipython/ipython
- Pillow (https://github.com/python-pillow/Pillow)
    - 7 webbadges:
        - https://travis-ci.org/python-pillow/Pillow
        - https://travis-ci.org/python-pillow/pillow-wheels
        - https://ci.appveyor.com/project/Pythonpillow/pillow
        - https://pypi.python.org/pypi/Pillow/
        - https://pypi.python.org/pypi/Pillow/
        - https://coveralls.io/r/python-pillow/Pillow?branch=master
        - https://landscape.io/github/python-pillow/Pillow/master
- Pypot (https://github.com/poppy-project/pypot)
    - Le readme semble bien écrit...
    - 2 webbadges:
        - https://travis-ci.org/poppy-project/pypot
        - https://camo.githubusercontent.com/57d97641b5dafc50c0bd24f2afad9973d9608e7e/68747470733a2f2f7
- Mayavi (https://github.com/enthought/mayavi)
    - Le readme semble bien écrit...
    - 2 webbadges:
        - https://travis-ci.org/enthought/mayavi
        - http://codecov.io/github/enthought/mayavi?branch=master
- PEP8 (https://github.com/PyCQA/pep8)
    - Le readme semble bien écrit...
    - 1 webbadge:
        - https://travis-ci.org/PyCQA/pep8

6.1 PEP8

- https://github.com/PyCQA/pep8
- http://pep8.readthedocs.org/en/latest/

7 Glossaire

- autodoc
- docstring
- doctest
- docutils
- epydoc
- napoleon
- pandoc
- pydoc
- readthedocs
- restructuredtext
- sphinx
- unittest

8 TODO

Pages à lire:

- https://tom-christie.github.io/articles/pypi/
- http://write-the-docs.readthedocs.io/guide/
- Le livre de Tarek Ziadé: *Python, petit guide à l'usage du développeur agile*
- http://stackoverflow.com/questions/2557110/what-to-put-in-a-python-module-docstring.

9 Licence