AST1501 - Introduction to Research

Jo Bovy

Documenting your code

Basics of documentation

- Good documentation is essential for allowing people to use your code
- Should be as complete and up-to-date as possible
- Best if written along-side code development, don’t leave writing the documentation to *a later time*
- Document functions/classes/methods *and* have a guide to using your code

What goes in the documentation?

- *Installation guide*: list dependencies and how to install them, different ways to install your code (`pip` or `conda` or `...`)
- *Quick-start guide and tutorials*: giving examples of your code's use to help users get started
  - Good place to show off what your code can do!
- *Application programming interface (API)*: complete listing of all of your code's functions/classes/methods
  - Use automation to create this

docstrings

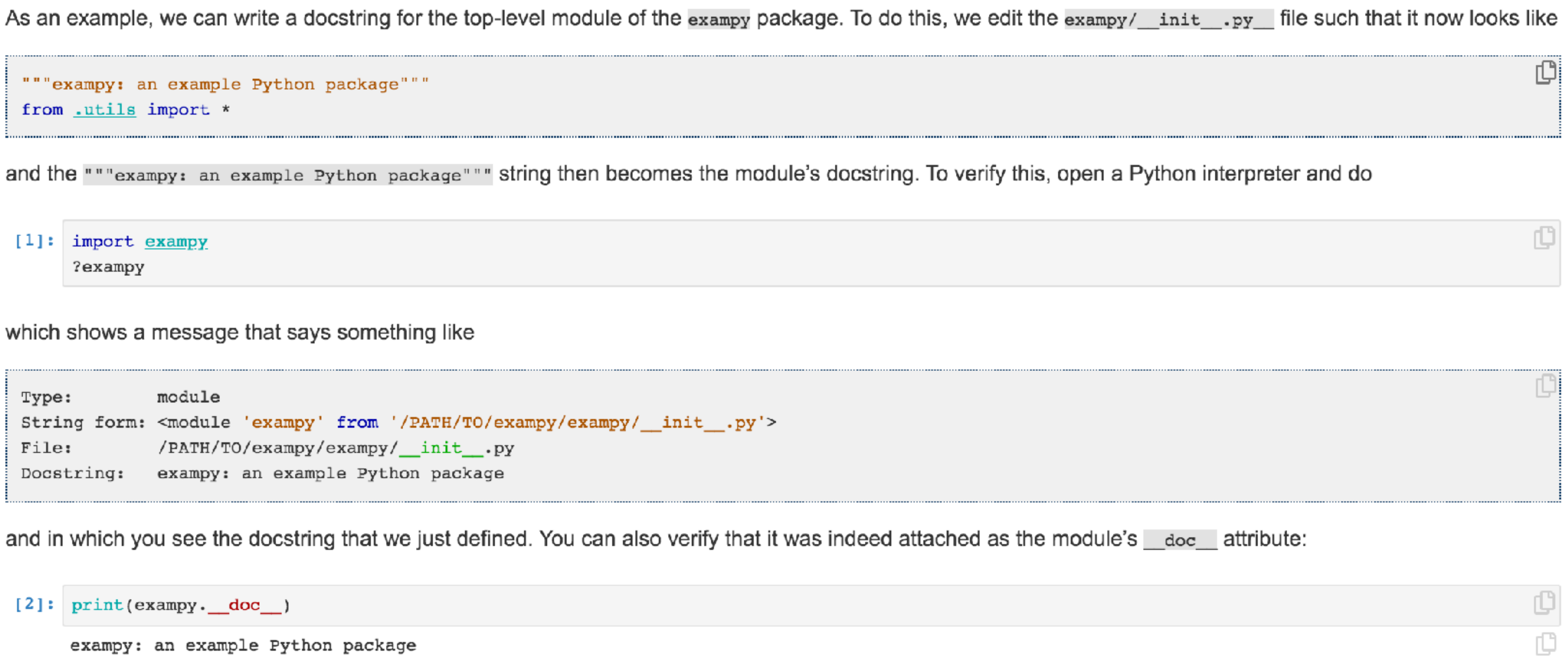
Python docstrings

- docstrings: built-in Python feature to document (sub)modules, functions, classes, and methods
- Place to put documentation on your code's *use*, not on its implementation (docstrings are for *users*, not for *developers*)
- Typically, multi-line strings enclosed in “““   ”””

Docstring placement

- To be automatically attached to the module, function, class, or method, put docstrings
  - Modules: Right at the beginning of the file
  - Functions: Right after the `def a_function(…):` statement
  - Classes: Right after the `class a_class:` statement
  - Methods: Right after the `def a_method(self,…):` statement
- Then automatically bound to the module/function/class/method’s `__doc__` attribute; you can also directly set this attribute!

Module docstring example



As an example, we can write a docstring for the top-level module of the `exampy` package. To do this, we edit the `exampy/__init__.py` file such that it now looks like

```
"""exampy: an example Python package"""
from .utils import *
```

and the `"""exampy: an example Python package"""` string then becomes the module's docstring. To verify this, open a Python interpreter and do

```
[1]: import exampy
     ?exampy
```

which shows a message that says something like

```
Type:        module
String form: <module 'exampy' from '/PATH/TO/exampy/exampy/__init__.py'>
File:        /PATH/TO/exampy/exampy/__init__.py
Docstring:   exampy: an example Python package
```

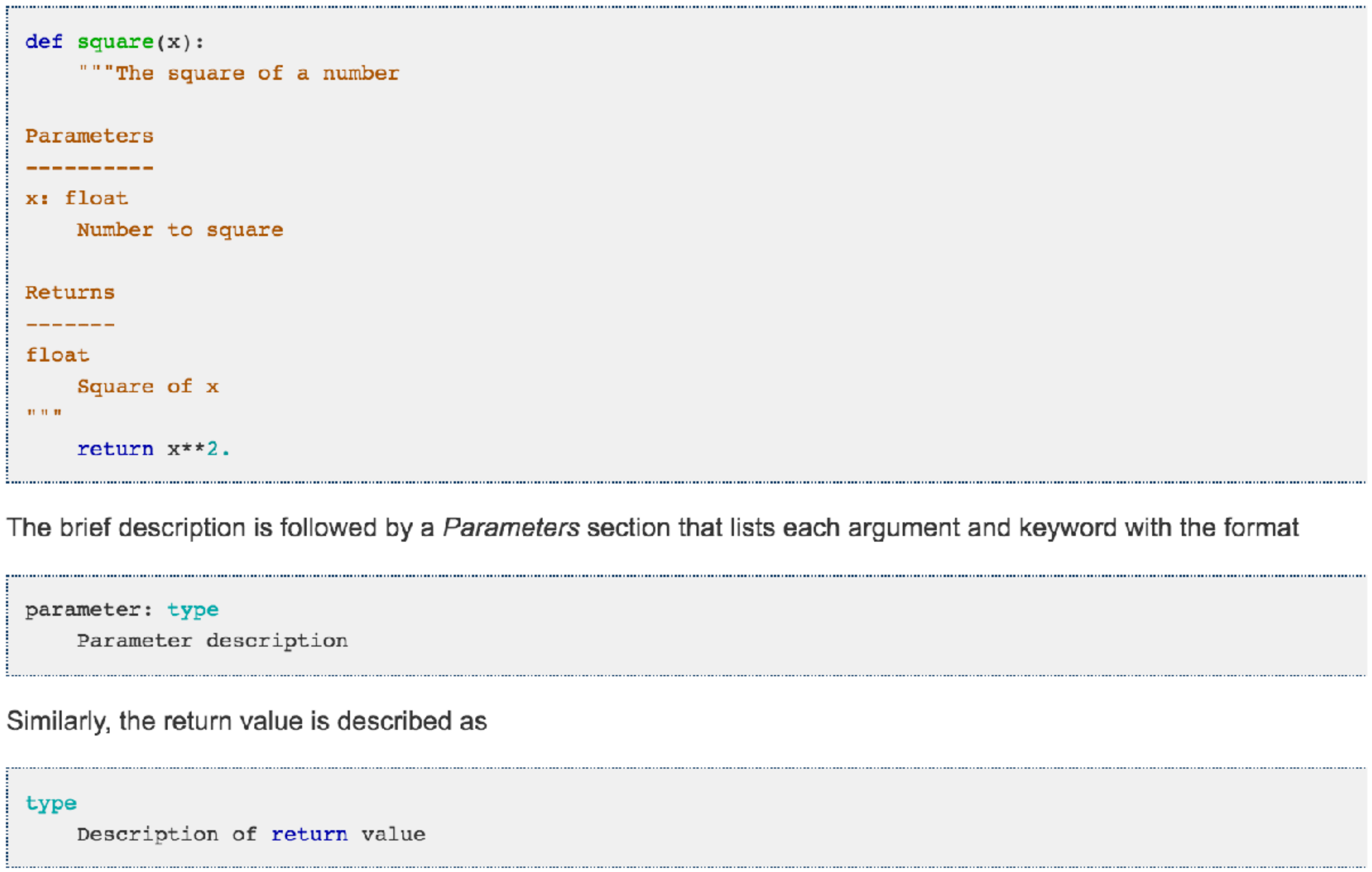
and in which you see the docstring that we just defined. You can also verify that it was indeed attached as the module's `__doc__` attribute:

```
[2]: print(exampy.__doc__)

     exampy: an example Python package
```

Functions and methods

- *Always* need multi-line docstrings,
  - give overview of what the function/method does
  - List input arguments and keywords
  - List outputs
- For methods, we don't document `self` (because assumed and always the same), so methods are essentially the same as functions
- Follow a consistent style for all the docstrings in your code, e.g., the numpy docstring style

numpy-style docstrings



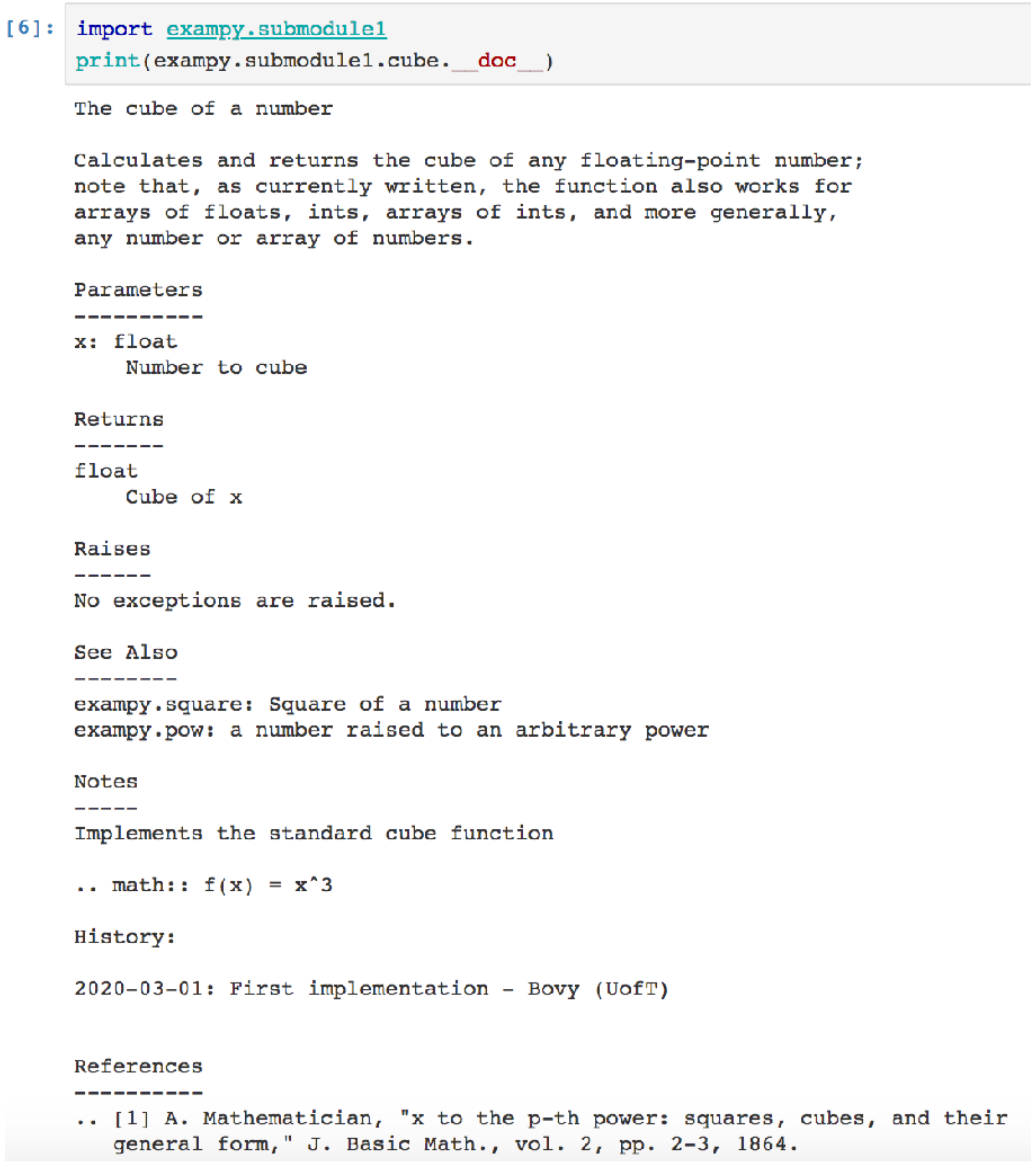
```python
def square(x):
    """The square of a number

Parameters
----------
x: float
    Number to square

Returns
-------
float
    Square of x
"""
    return x**2.
```

The brief description is followed by a *Parameters* section that lists each argument and keyword with the format

```
parameter: type
    Parameter description
```

Similarly, the return value is described as

```
type
    Description of return value
```



```
[6]: import exampy.submodule1
     print(exampy.submodule1.cube.__doc__)
```

```
The cube of a number

Calculates and returns the cube of any floating-point number;
note that, as currently written, the function also works for
arrays of floats, ints, arrays of ints, and more generally,
any number or array of numbers.

Parameters
----------
x: float
    Number to cube

Returns
-------
float
    Cube of x

Raises
------
No exceptions are raised.

See Also
--------
exampy.square: Square of a number
exampy.pow: a number raised to an arbitrary power

Notes
-----
Implements the standard cube function

.. math:: f(x) = x^3

History:

2020-03-01: First implementation - Bovy (UofT)


References
----------
.. [1] A. Mathematician, "x to the p-th power: squares, cubes, and their
   general form," J. Basic Math., vol. 2, pp. 2-3, 1864.
```

Ways to document your code

Ways to document your code

- *Always* document code itself functions, classes, etc.
- For more general documentation (installation, dependencies, example use, how-tos):
    - Use the GitHub ReadMe: Markdown is powerful and allows you to easily create a small documentation site
    - Stand-alone documentation site: sphinx

sphinx

What is `sphinx`?

- Python tool to typeset documentation from a set of reStructuredText files, with a lot of support for documentation tools
- reStructuredText: simple markup language for text documents that can be turned into HTML, LaTeX, …
- `pip install sphinx`

Getting started with `sphinx`

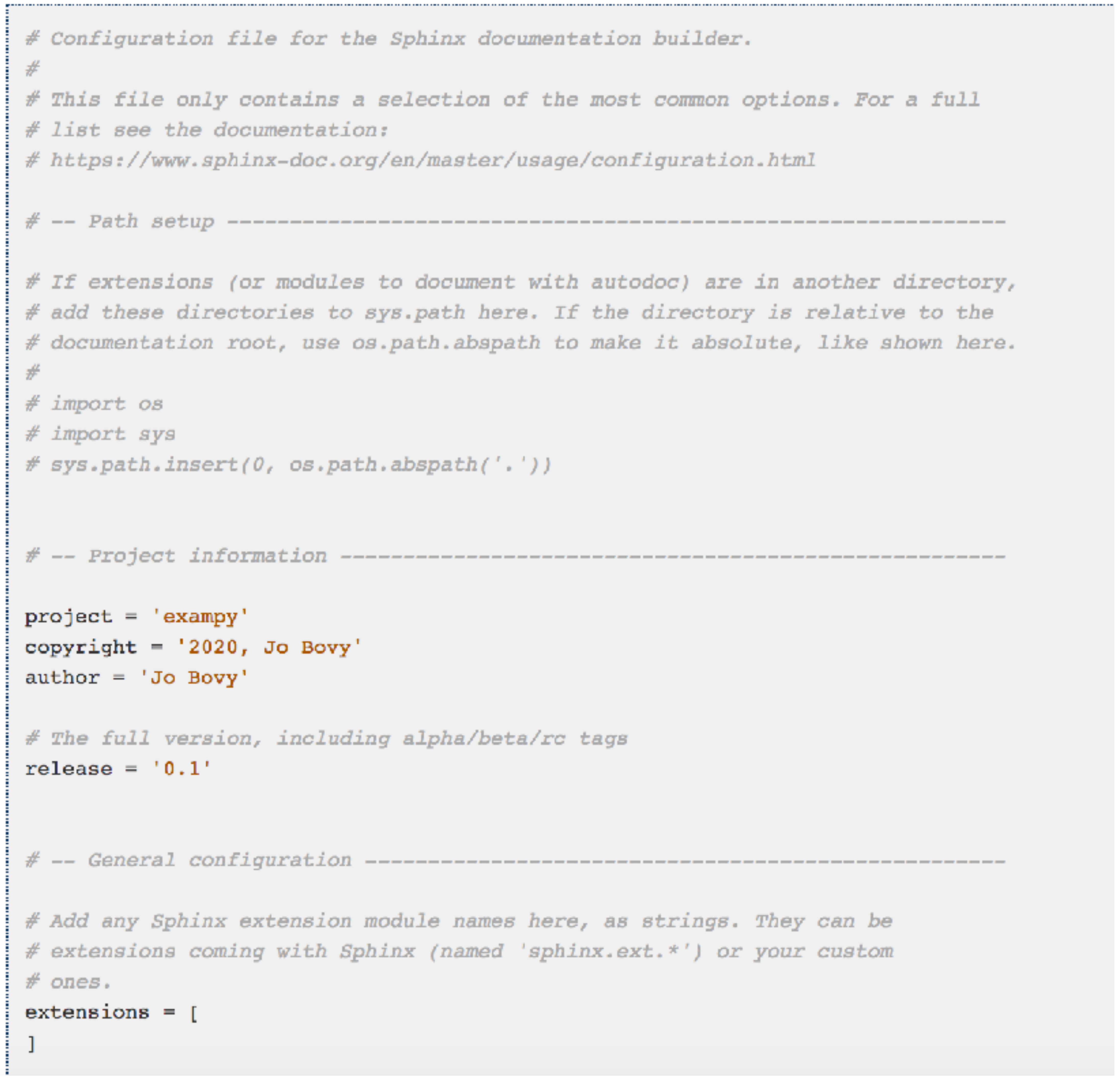
- Start a directory `doc/` or `docs/`
- In that directory type `sphinx-quickstart`
- Answer a few questions
    - Name of the package
    - Author
    - Version
    - Separate `build/` and `source/` directories (otherwise have `_build/` in `source/`): yes, a good idea!
- After this, you have the basic outline of your documentation

```
build/
source/
    _static/
    _templates/
    conf.py
    index.rst
Makefile
make.bat
```

`conf.py`

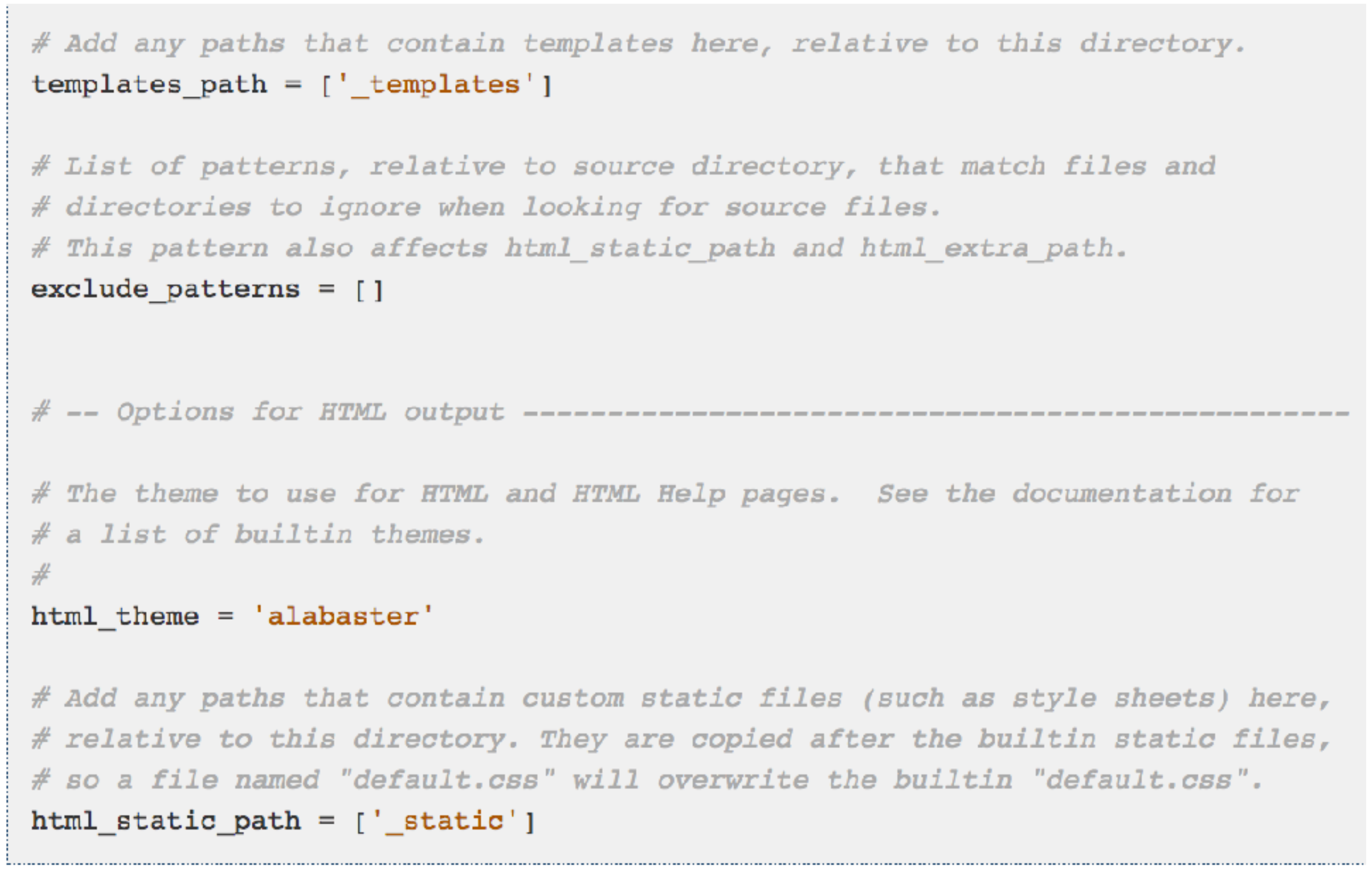
- Configuration file, as a Python script (executed, so can contain Python code)
- Used to set all of the configuration:
  - General: name, author, version, extensions to use, general configuration parameters
- Configuration parameters for different output types: HTML, LaTeX, …

Starting `conf.py`



```
# Configuration file for the Sphinx documentation builder.
#
# This file only contains a selection of the most common options. For a full
# list see the documentation:
# https://www.sphinx-doc.org/en/master/usage/configuration.html

# -- Path setup --------------------------------------------------------------

# If extensions (or modules to document with autodoc) are in another directory,
# add these directories to sys.path here. If the directory is relative to the
# documentation root, use os.path.abspath to make it absolute, like shown here.
#
# import os
# import sys
# sys.path.insert(0, os.path.abspath('.'))


# -- Project information -----------------------------------------------------

project = 'exampy'
copyright = '2020, Jo Bovy'
author = 'Jo Bovy'

# The full version, including alpha/beta/rc tags
release = '0.1'


# -- General configuration ---------------------------------------------------

# Add any Sphinx extension module names here, as strings. They can be
# extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
# ones.
extensions = [
]
```

Starting `conf.py` (continued)

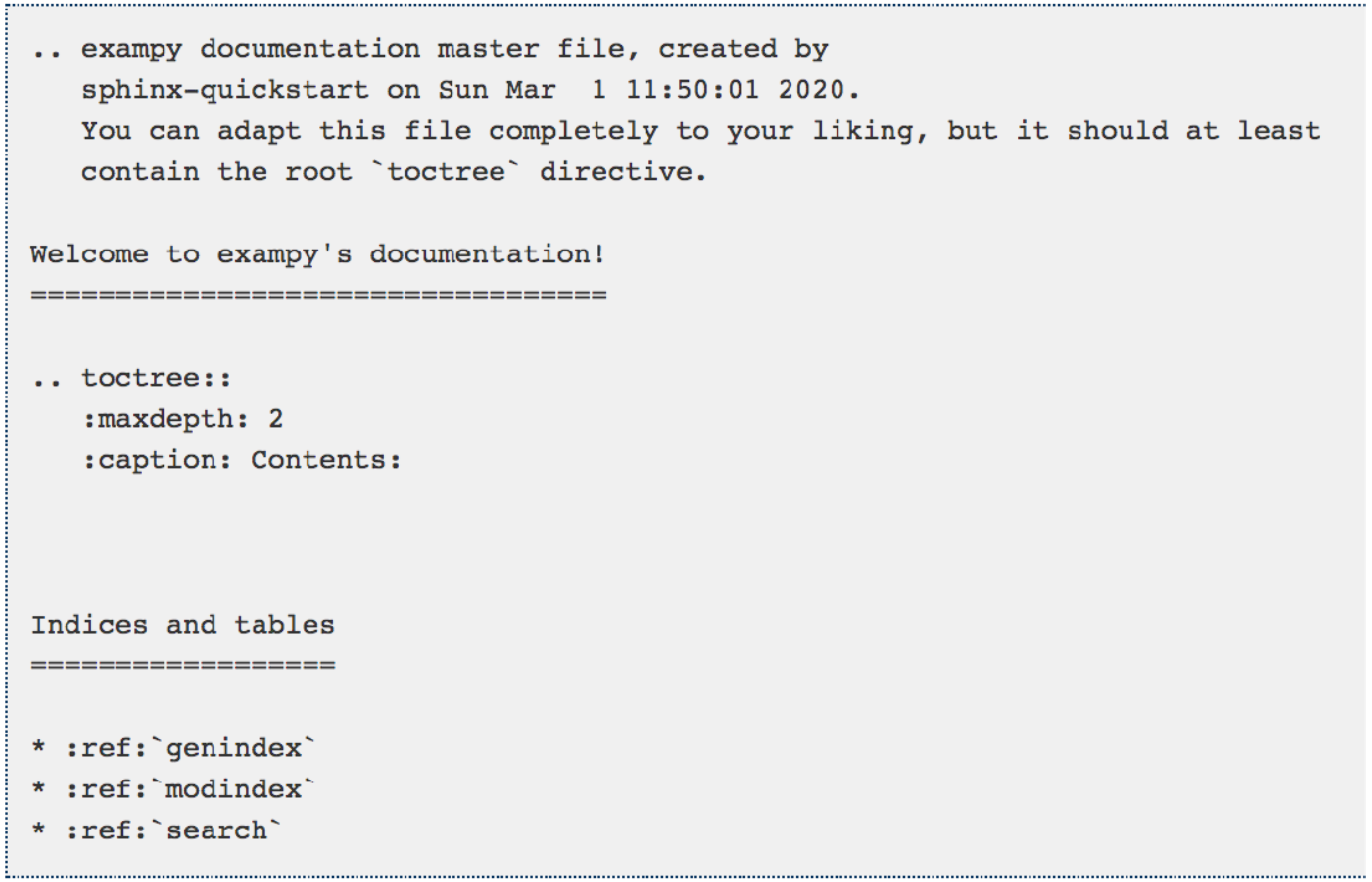


```python
# Add any paths that contain templates here, relative to this directory.
templates_path = ['_templates']

# List of patterns, relative to source directory, that match files and
# directories to ignore when looking for source files.
# This pattern also affects html_static_path and html_extra_path.
exclude_patterns = []


# -- Options for HTML output -------------------------------------------------

# The theme to use for HTML and HTML Help pages.  See the documentation for
# a list of builtin themes.
#
html_theme = 'alabaster'

# Add any paths that contain custom static files (such as style sheets) here,
# relative to this directory. They are copied after the builtin static files,
# so a file named "default.css" will overwrite the builtin "default.css".
html_static_path = ['_static']
```

Documentation pages

- A set of `.rst` files in reStructuredText format
- `index.rst` contains the main “toctree”, a table of contents
  - Only files listed in this toctree or in toctrees in those files (etc.) are included in the documentation
  - toctree is an example of a *directive*, a way of telling sphinx (and rst) about different elements (e.g., math, images, …)
- `index.rst` can contain more, but the main toctree is essential

Starting `index.rst`



```rst
.. exampy documentation master file, created by
   sphinx-quickstart on Sun Mar  1 11:50:01 2020.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Welcome to exampy's documentation!
==================================

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



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

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

Generating the documentation

- Use the provided Makefile, type `make` for help

```
make
```

gives

```
Sphinx v2.2.0
Please use `make target' where target is one of
  html        to make standalone HTML files
  dirhtml     to make HTML files named index.html in directories
  singlehtml  to make a single large HTML file
  pickle      to make pickle files
  json        to make JSON files
  htmlhelp    to make HTML files and an HTML help project
  qthelp      to make HTML files and a qthelp project
  devhelp     to make HTML files and a Devhelp project
  epub        to make an epub
  latex       to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  latexpdf    to make LaTeX and PDF files (default pdflatex)
  latexpdfja  to make LaTeX files and run them through platex/dvipdfmx
  text        to make text files
  man         to make manual pages
  texinfo     to make Texinfo files
  info        to make Texinfo files and run them through makeinfo
  gettext     to make PO message catalogs
  changes     to make an overview of all changed/added/deprecated items
  xml         to make Docutils-native XML files
  pseudoxml   to make pseudoxml-XML files for display purposes
  linkcheck   to check all external links for integrity
  doctest     to run all doctests embedded in the documentation (if enabled)
  coverage    to run coverage check of the documentation (if enabled)
```

```
make
```

gives

```
Sphinx v2.2.0
Please use `make target' where target is one of
  html        to make standalone HTML files
  dirhtml     to make HTML files named index.html in directories
  singlehtml  to make a single large HTML file
  pickle      to make pickle files
  json        to make JSON files
  htmlhelp    to make HTML files and an HTML help project
  qthelp      to make HTML files and a qthelp project
  devhelp     to make HTML files and a Devhelp project
  epub        to make an epub
  latex       to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  latexpdf    to make LaTeX and PDF files (default pdflatex)
  latexpdfja  to make LaTeX files and run them through platex/dvipdfmx
  text        to make text files
  man         to make manual pages
  texinfo     to make Texinfo files
  info        to make Texinfo files and run them through makeinfo
  gettext     to make PO message catalogs
  changes     to make an overview of all changed/added/deprecated items
  xml         to make Docutils-native XML files
  pseudoxml   to make pseudoxml-XML files for display purposes
  linkcheck   to check all external links for integrity
  doctest     to run all doctests embedded in the documentation (if enabled)
  coverage    to run coverage check of the documentation (if enabled)
```

```
make
```

gives

```
Sphinx v2.2.0
Please use `make target' where target is one of
  html         to make standalone HTML files
  dirhtml      to make HTML files named index.html in directories
  singlehtml   to make a single large HTML file
  pickle       to make pickle files
  json         to make JSON files
  htmlhelp     to make HTML files and an HTML help project
  qthelp       to make HTML files and a qthelp project
  devhelp      to make HTML files and a Devhelp project
  epub         to make an epub
  latex        to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  latexpdf     to make LaTeX and PDF files (default pdflatex)
  latexpdfja   to make LaTeX files and run them through platex/dvipdfmx
  text         to make text files
  man          to make manual pages
  texinfo      to make Texinfo files
  info         to make Texinfo files and run them through makeinfo
  gettext      to make PO message catalogs
  changes      to make an overview of all changed/added/deprecated items
  xml          to make Docutils-native XML files
  pseudoxml    to make pseudoxml-XML files for display purposes
  linkcheck    to check all external links for integrity
  doctest      to run all doctests embedded in the documentation (if enabled)
  coverage     to run coverage check of the documentation (if enabled)
```

```
make
```

gives

```
Sphinx v2.2.0
Please use `make target' where target is one of
  html         to make standalone HTML files
  dirhtml      to make HTML files named index.html in directories
  singlehtml   to make a single large HTML file
  pickle       to make pickle files
  json         to make JSON files
  htmlhelp     to make HTML files and an HTML help project
  qthelp       to make HTML files and a qthelp project
  devhelp      to make HTML files and a Devhelp project
  epub         to make an epub
  latex        to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  latexpdf     to make LaTeX and PDF files (default pdflatex)
  latexpdfja   to make LaTeX files and run them through platex/dvipdfmx
  text         to make text files
  man          to make manual pages
  texinfo      to make Texinfo files
  info         to make Texinfo files and run them through makeinfo
  gettext      to make PO message catalogs
  changes      to make an overview of all changed/added/deprecated items
  xml          to make Docutils-native XML files
  pseudoxml    to make pseudoxml-XML files for display purposes
  linkcheck    to check all external links for integrity
  doctest      to run all doctests embedded in the documentation (if enabled)
  coverage     to run coverage check of the documentation (if enabled)
```

Example `.rst` files

```
Installation instructions
=========================

Dependencies
------------

``exampy`` does not have any dependencies.

Installation
------------

``exampy`` is currently not yet available on PyPI, but it can be
installed by downloading the source code or cloning the GitHub
repository and running the standard::

    python setup.py install

command or its usual variants (``python setup.py install --user``,
``python setup.py install --prefix=/PATH/TO/INSTALL/DIRECTORY``,
etc.).

For more info, please open an Issue on the GitHub page.
```

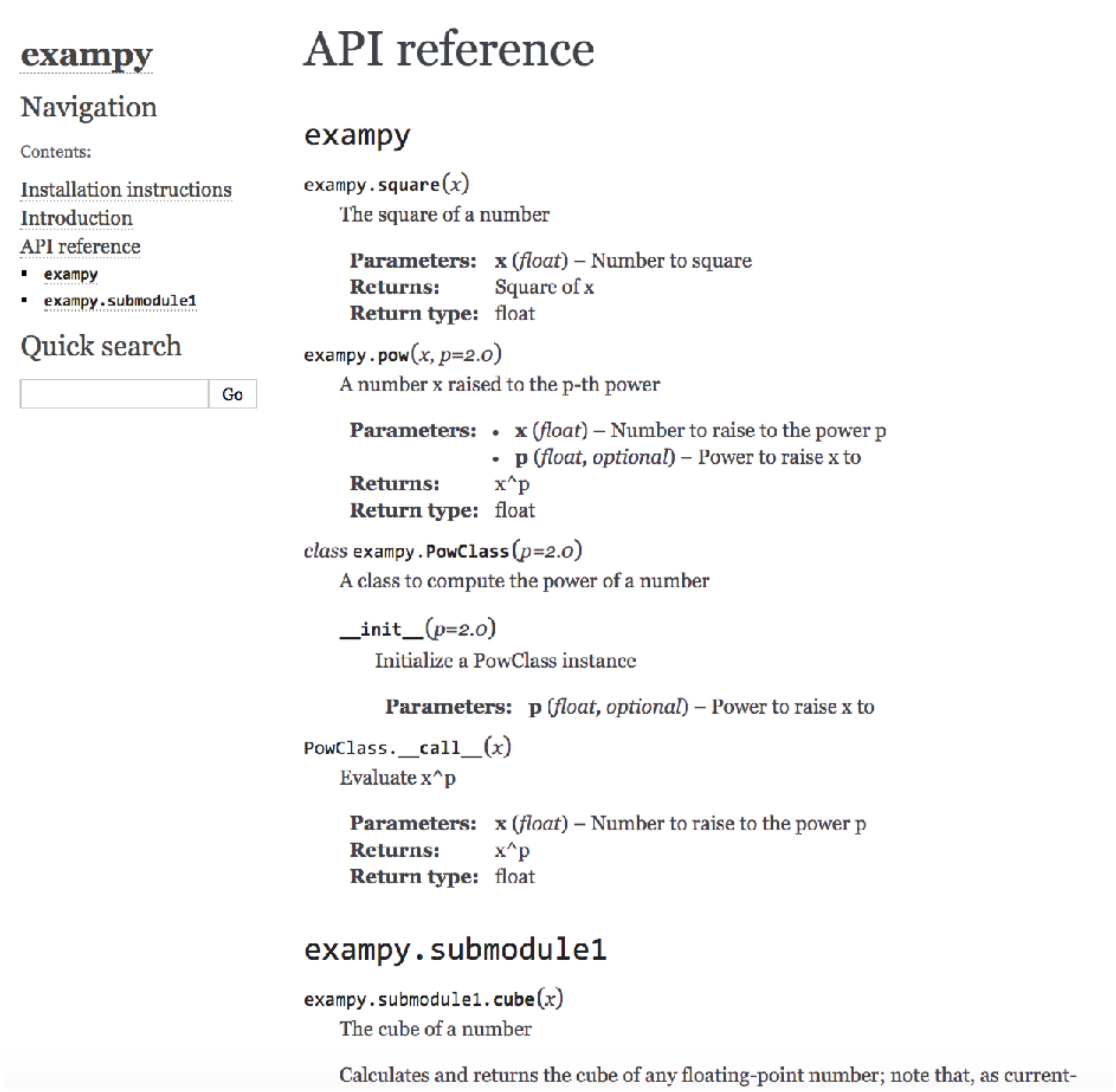
```
Introduction
============

``exampy`` is an example Python package that contains some very basic math
functions. As an example, we can compute the square of a number as::

       >>> import exampy
       >>> exampy.square(3.)
       # 9.

Similarly, we can compute the cube of a number, but this functionality is part
of the ``exampy.submodule1`` submodule:

.. code-block:: python

   >>> import exampy.submodule1
   >>> exampy.submodule1.cube(3.)
   # 27.

A general power function ``pow`` is included at the top-level, for example, to
get the fourth power of 3, do::

    >>> exampy.pow(3.,p=4.)
    # 81.

This concludes the discussion of all of ``exampy``'s basic
functionality.
```

Including docstrings

- sphinx has a built-in extension to grab docstrings from the code and insert them into the documentation (e.g., when creating the API)
- Extension: `autodoc` (add “sphinx.ext.autodoc” to the extensions list in `conf.py`)
  - Also use `napoleon` for parsing numpy-style docstrings ”sphinx.ext.napoleon”
- Three main directives:
  - `.. autofunction:: func`
  - `.. autoclass:: a_class`
    - Also has the `:members:` option to list member methods to include
  - `.. automethod:: a_method`

Example usage

```
API reference
=============

``exampy``
----------

.. autofunction:: exampy.square

.. autofunction:: exampy.pow

.. autoclass:: exampy.PowClass
   :members: __init__

.. automethod:: exampy.PowClass.__call__

``exampy.submodule1``
---------------------

.. autofunction:: exampy.submodule1.cube
```



exampy

Navigation

Contents:

Installation instructions
Introduction
API reference
- `exampy`
- `exampy.submodule1`

Quick search

API reference

`exampy`

`exampy.square`(*x*)

The square of a number

Parameters: **x** (*float*) – Number to square
Returns: Square of x
Return type: float

`exampy.pow`(*x*, *p=2.0*)

A number x raised to the p-th power

Parameters:
- **x** (*float*) – Number to raise to the power p
- **p** (*float, optional*) – Power to raise x to

Returns: x^p
Return type: float

class `exampy.PowClass`(*p=2.0*)

A class to compute the power of a number

`__init__`(*p=2.0*)

Initialize a PowClass instance

Parameters: **p** (*float, optional*) – Power to raise x to

`PowClass.__call__`(*x*)

Evaluate x^p

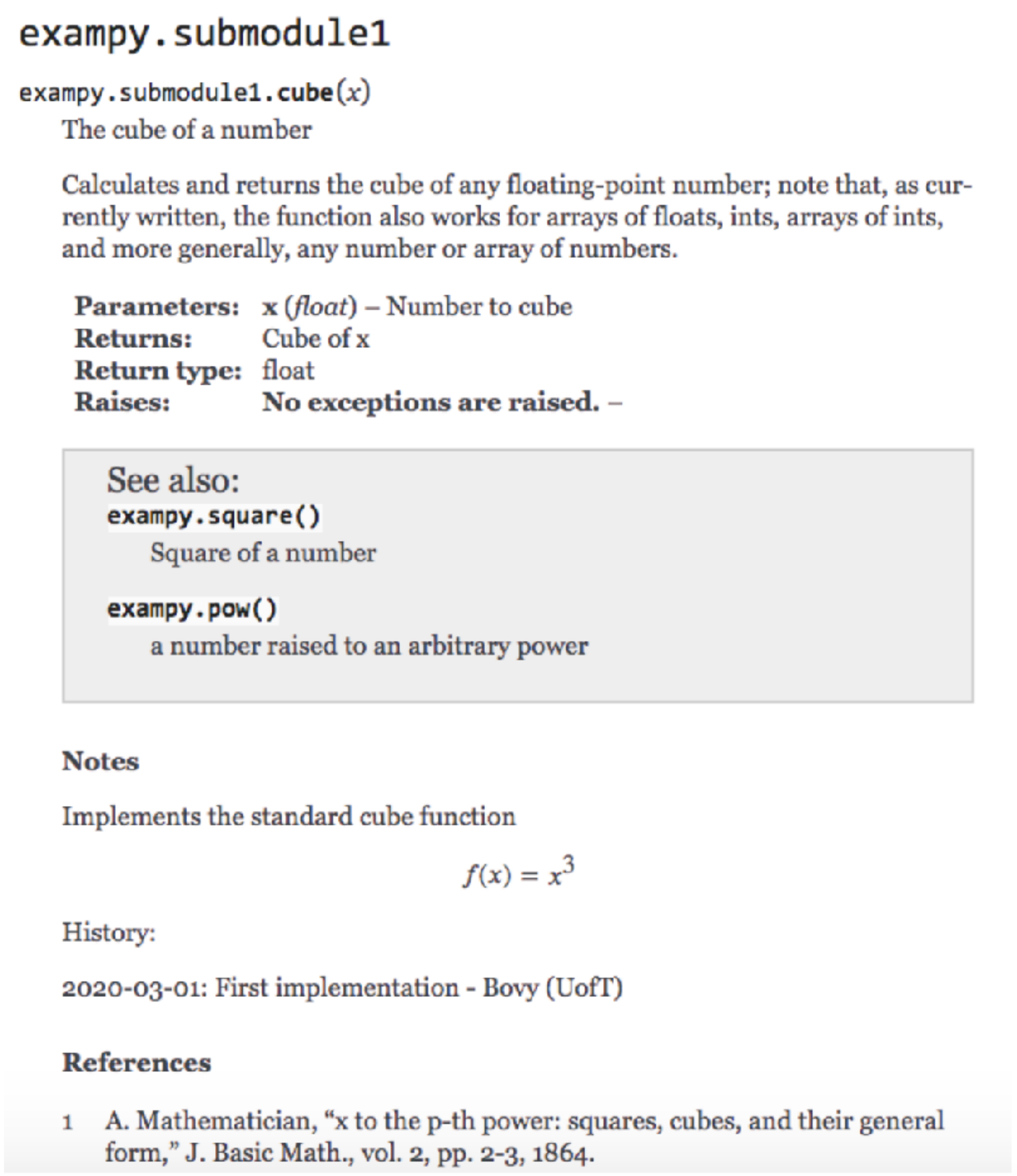
Parameters: **x** (*float*) – Number to raise to the power p
Returns: x^p
Return type: float

`exampy.submodule1`

`exampy.submodule1.cube`(*x*)

The cube of a number

Calculates and returns the cube of any floating-point number; note that, as current-



`exampy.submodule1`

`exampy.submodule1.cube`(*x*)

The cube of a number

Calculates and returns the cube of any floating-point number; note that, as currently written, the function also works for arrays of floats, ints, arrays of ints, and more generally, any number or array of numbers.

Parameters:	**x** (*float*) – Number to cube
Returns:	Cube of x
Return type:	float
Raises:	**No exceptions are raised.** –

See also:

`exampy.square()`
Square of a number

`exampy.pow()`
a number raised to an arbitrary power

Notes

Implements the standard cube function

$$f(x) = x^3$$

History:

2020-03-01: First implementation - Bovy (UofT)

References

1 A. Mathematician, “x to the p-th power: squares, cubes, and their general form,” J. Basic Math., vol. 2, pp. 2-3, 1864.

Using `jupyter notebooks` in `sphinx` documentation

- Easy to write combination of text and code in jupyter notebooks, and to include figures
- Extensions: `nbsphinx` to include jupyter notebooks *as they are* in sphinx documentation
    - `python3 -m pip install nbsphinx`
    - Add “nbsphinx” to the extensions list in `conf.py`
- Then can just add notebook in a toctree!