default$_{values}$

Release 0.0.8

Dominic Davis-Foster

Aug 26, 2020

CONTENTS

Sphinx extension to show default values in documentation.

Docs	
Tests	
PyPI	
Activity	
Other	

CHAPTER

ONE

INSTALLATION

from PyPI

```
$ python3 -m pip install default_values --user
```

from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/default_values@master --user
```

1.1 Usage

Enable `default_values` by adding "sphinxcontrib.default_values" to the `extensions` variable in `conf.py`:

```
extensions = [
    ...
    "sphinxcontrib.default_values",
    ]
```

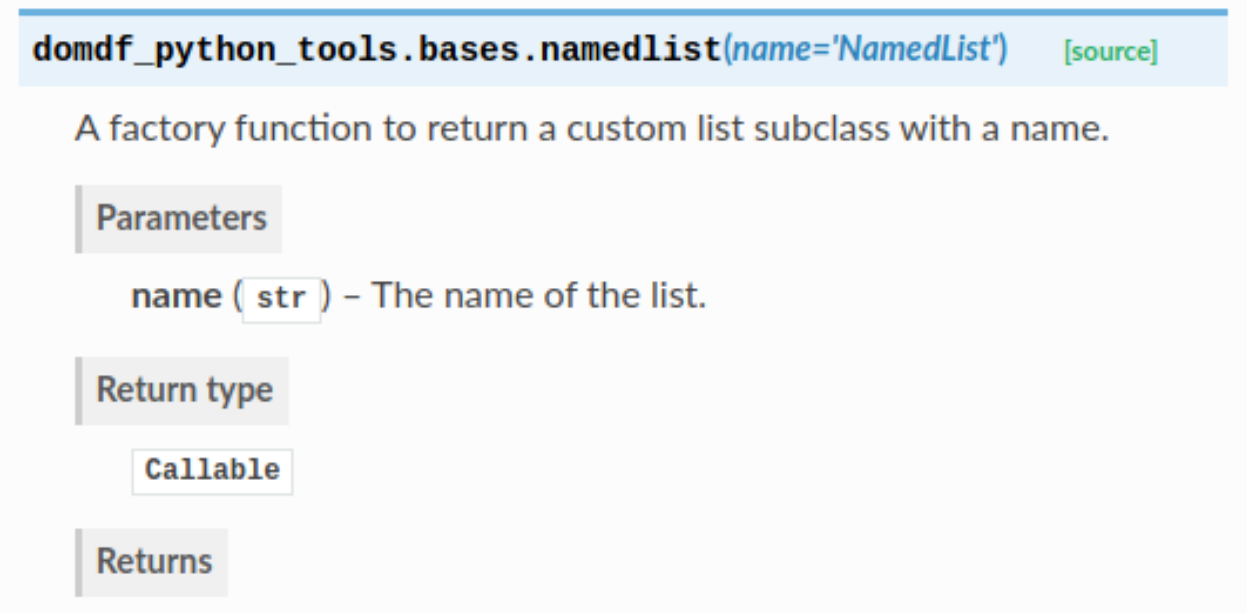
For more information see https://www.sphinx-doc.org/en/master/usage/extensions/index.html#third-party-extensions .

This extension shows the default values in autodoc-formatted docstrings.

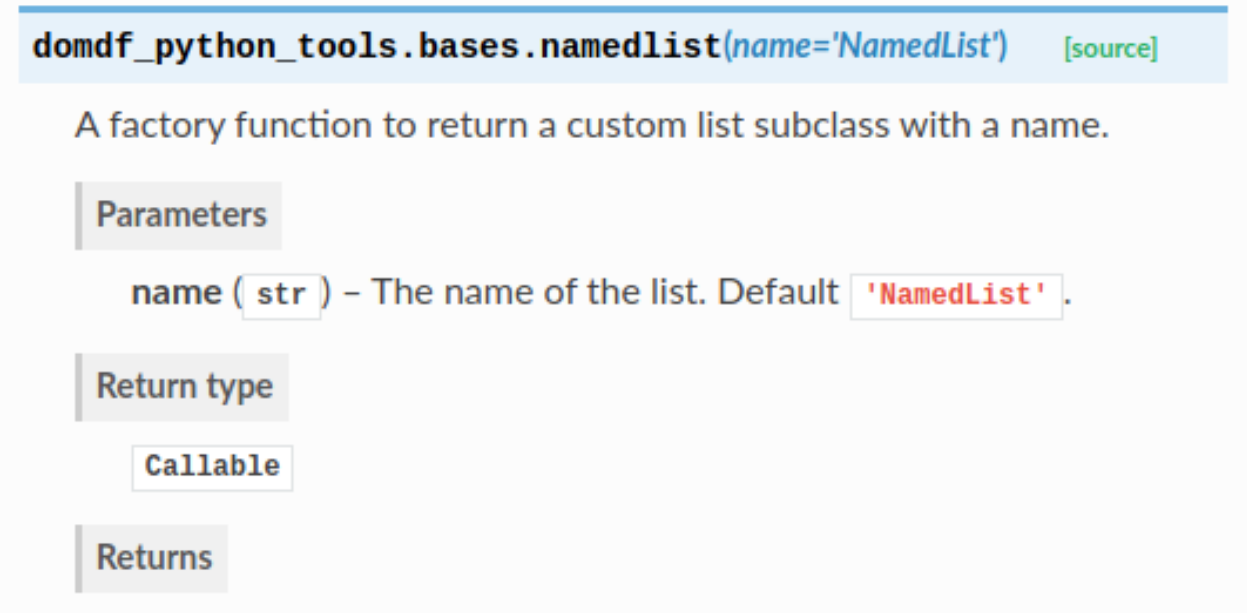
The default behaviour of autodoc is to turn this:

```
def namedlist(name: str = "NamedList") -> Callable:
    """
    A factory function to return a custom list subclass with a name.

    :param name: The name of the list.
    :default name: :py:obj:`True`

    :return:
    """
```

into this:



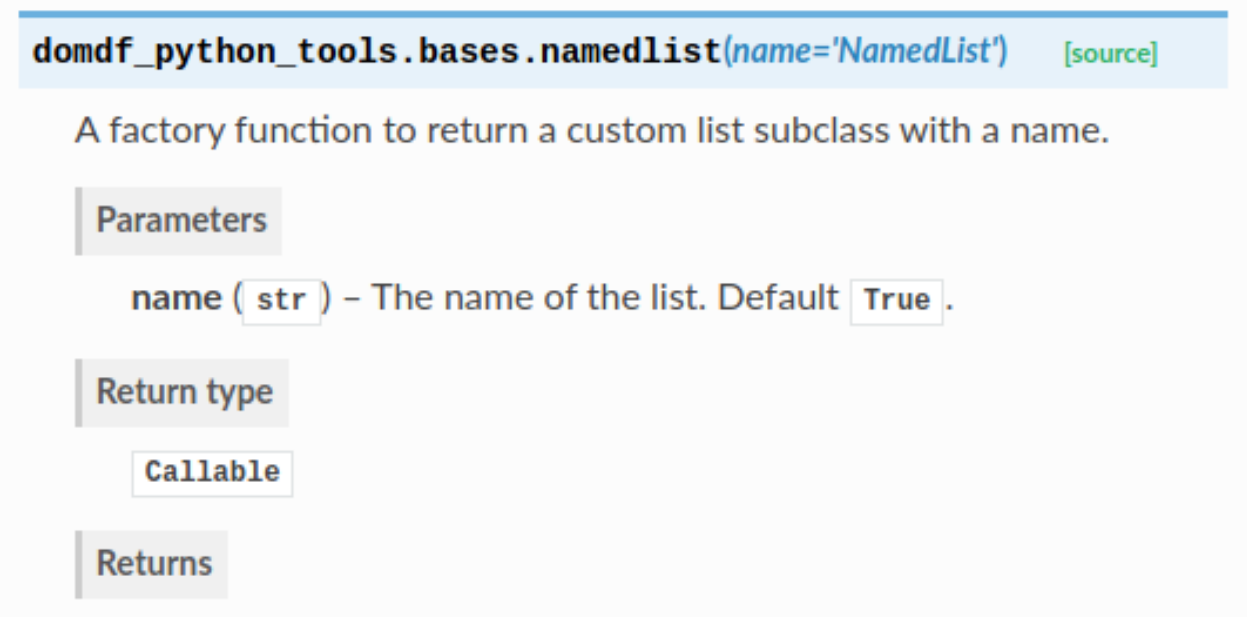
With `default_values` enabled, the documentation will now look like this:



Default values are taken from the function/class signature. They can be overridden using the `:default <argname>: <default value>` option in the docstring:

```
:param name: The name of the list.
:default name: :py:obj:`True`
```

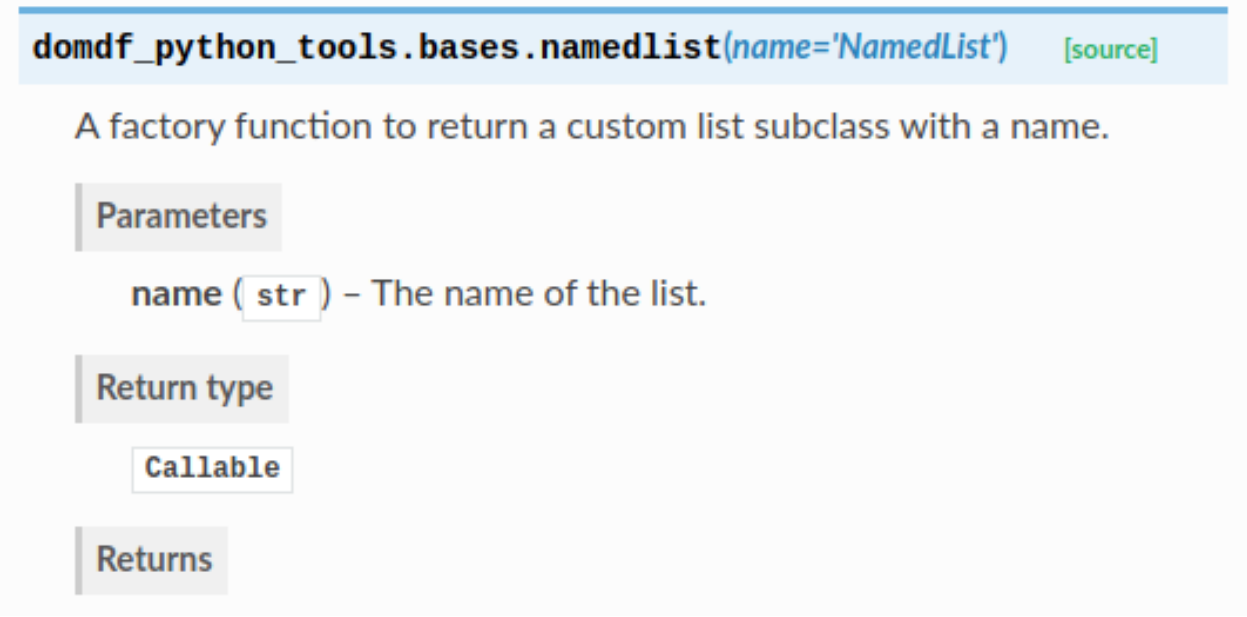
which will produce:



The value must be formatted how you would like it to be displayed in Sphinx. This can be useful when the default value in the signature is `None` and the true default value is assigned in the function body, such as for a mutable default argument.

The default value can be suppressed using the `:no-default <argname>` option:

```
:param name: The name of the list.
:no-default name:
```



This allows for default values to be suppressed on a per-argument basis.

No default value is shown if the argument does not have a default value.

The formatting of the default value can be customised using the `default_description_format` option in `setup.py`. By default this is `"Default %s"`.

1.2 API Reference

A Sphinx directive to specify that a module has extra requirements, and show how to install them.

Functions:

process_docstring(app, what, name, obj, ...)	Add default values to the docstring.
process_default_format(app)	Prepare the formatting of the default value.
setup(app)	Setup Sphinx Extension.

sphinxcontrib.default_values.**process_default_format**(*app*)
Prepare the formatting of the default value.

Parameters **app** (Sphinx) –

Return type None

sphinxcontrib.default_values.**process_docstring**(*app*, *what*, *name*, *obj*, *options*, *lines*)
Add default values to the docstring.

Parameters

- **app** (Sphinx) –
- **what** –
- **name** –
- **obj** –
- **options** –
- **lines** (List[str]) – List of strings representing the current contents of the docstring.

Return type None

sphinxcontrib.default_values.**setup**(*app*)
Setup Sphinx Extension.

Parameters **app** (Sphinx) –

Return type Dict[str, Any]

Returns

1.3 Demo

sphinxcontrib.default_values.demo.**demo**(*a*, *b=0.0*, *c=''*, *d=' '*, *e='hello world'*, *f=()*, *g=Decimal('12.34')*, *h=1234*, *i=None*, *j=None*, *k=None*, *l=''*)

Parameters

- **a** (Any) – No default.
- **b** (float) – A float. Default 0.0.
- **c** (str) – An empty string. Default ''.
- **d** (str) – A space (or a smiley face?). Default '␣'.
- **e** (str) – A string. Default 'hello␣world'.
- **f** (Tuple) – A Tuple. Default ().
- **g** (Decimal) – A Decimal. Default Decimal('12.34').
- **h** (int) – An int. Default 1234.
- **i** (Optional[List[str]]) – Default None. Default None.
- **j** (Optional[List[str]]) – Overridden default. Default [].
- **k** (Optional[List[str]]) – Suppressed default.
- **l** (str) – This is a really long description. It spans multiple lines. The quick brown fox jumps over the lazy dog. The default value should be added at the end regardless. Default ''.

The description for d lacked a fullstop at the end, but one was added automatically.

The above example was created from the following Python code:

```
# stdlib
from decimal import Decimal  # pragma: no cover
from typing import Any, List, Optional, Tuple  # pragma: no cover

__all__ = ["demo"]  # pragma: no cover


def demo(
        a: Any,
        b: float = 0.0,
        c: str = '',
        d: str = ' ',
        e: str = "hello world",
        f: Tuple = (),
        g: Decimal = Decimal("12.34"),
        h: int = 1234,
        i: Optional[List[str]] = None,
        j: Optional[List[str]] = None,
        k: Optional[List[str]] = None,
        l: str = '',
        ):  # pragma: no cover
    """

    :param a: No default.
```

(continues on next page)

(continued from previous page)

```
    :param b: A float.
    :param c: An empty string.
    :param d: A space (or a smiley face?)
    :param e: A string.
    :param f: A Tuple.
    :param g: A Decimal.
    :param h: An int.
    :param i: Default None.
    :param j: Overridden default.
    :default j: ``[]``
    :param k: Suppressed default.
    :no-default k:
    :param l: This is a really long description.
        It spans multiple lines.
        The quick brown fox jumps over the lazy dog.
        The default value should be added at the end regardless.


    The description for ``d`` lacked a fullstop at the end, but one was added␣
↪automatically.
    """
```

The **PEP 484** type hints were added by sphinx-autodoc-typehints.

1.4 Contributing

`default_values` uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

1.4.1 Coding style

Yapf is used for code formatting, and isort is used to sort imports.

`yapf` and `isort` can be run manually via `pre-commit`:

```
$ pre-commit run yapf -a
$ pre-commit run isort -a
```

The complete autoformatting suite can be run with `pre-commit`:

```
$ pre-commit run -a
```

1.4.2 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6, run:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

1.4.3 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

1.4.4 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

1.5 Downloading source code

The `default_values` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/default_values"

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/default_values"
> Cloning into 'default_values'...
> remote: Enumerating objects: 47, done.
> remote: Counting objects: 100% (47/47), done.
> remote: Compressing objects: 100% (41/41), done.
> remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
> Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
> Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

1.5.1 Building from source

The recommended way to build `default_values` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

View the Function Index or browse the Source Code.

Browse the GitHub Repository

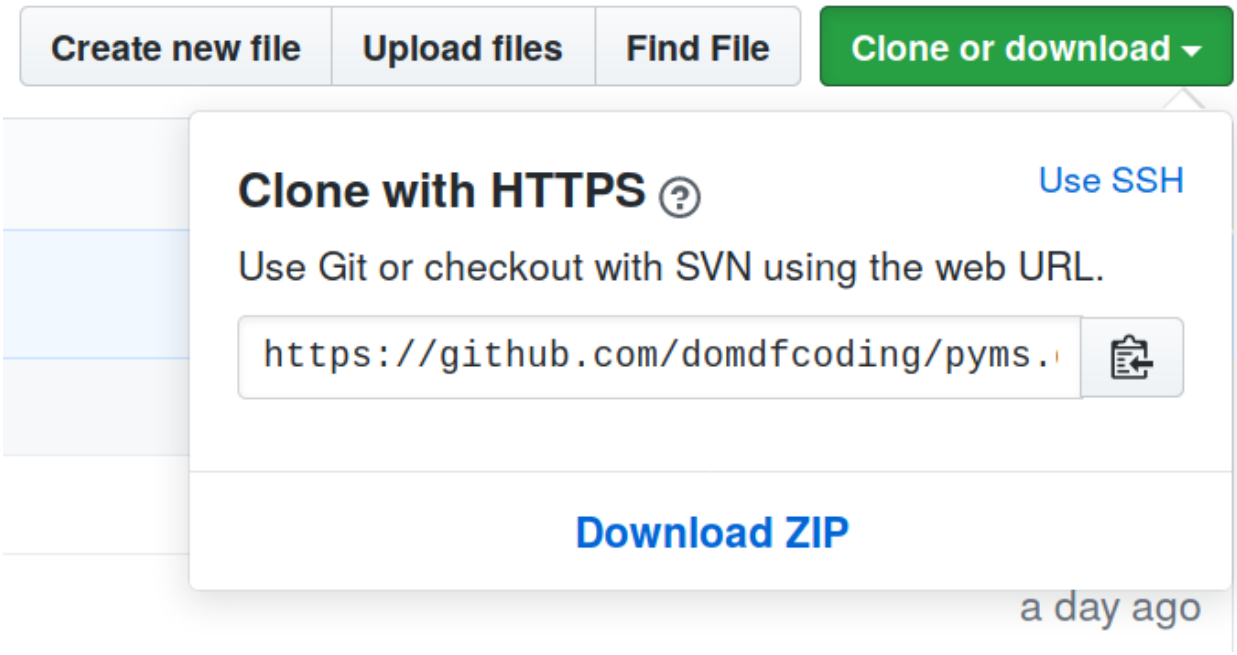


Fig. 1: Downloading a 'zip' file of the source code

PYTHON MODULE INDEX

S

INDEX