

msaSDK Module

`.utils.example_google_doc_style`

Example Google style docstrings.

This module demonstrates documentation as specified by the `Google Python Style Guide`. Docstrings may extend over multiple lines. Sections are created with a section header and a colon followed by a block of indented text.

Example

Examples can be given using either the `Example` or `Examples` sections. Sections support any `reStructuredText` formatting, including literal blocks::

```
$ python example_google.py
```

Section breaks are created by resuming unindented text. Section breaks are also implicitly created anytime a new section starts.

ATTRIBUTE	DESCRIPTION
<code>module_level_variable1</code>	<p>Module level variables may be documented in either the <code>Attributes</code> section of the module docstring, or in an inline docstring immediately following the variable.</p> <p>Either form is acceptable, but the two should not be mixed. Choose one convention to document module level variables and be consistent with it.</p> <p>TYPE: <code>int</code></p>

Todo

- For module TODOs
- You have to also use `sphinx.ext.todo` extension

.. _Google Python Style Guide: <http://google.github.io/styleguide/pyguide.html>

Attributes

module_level_variable1 module-attribute

```
module_level_variable1 = 12345
```

module_level_variable2 module-attribute

```
module_level_variable2 = 98765
```

int: Module level variable documented inline.

The docstring may span multiple lines. The type may optionally be specified on the first line, separated by a colon.

Classes

ExampleClass

Bases: object

The summary line for a class docstring should fit on one line.

If the class has public attributes, they may be documented here in an Attributes section and follow the same formatting as a function's Args section. Alternatively, attributes may be documented inline with the attribute's declaration (see **init** method below).

Properties created with the @property decorator should be documented in the property's getter method.

ATTRIBUTE	DESCRIPTION
<small>attr1</small>	Description of <small>attr1</small> . TYPE: <small>str</small>
<small>attr2</small>	obj: <small>int</small> , optional): Description of <small>attr2</small> .

Attributes

attr1 instance-attribute

```
attr1 = param1
```

attr2 instance-attribute

```
attr2 = param2
```

attr3 instance-attribute

```
attr3 = param3
```

attr4 instance-attribute

```
attr4 = ['attr4']
```

attr5 instance-attribute

```
attr5 = None
```

str: Docstring *after* attribute, with type specified.

Functions

`__init__`

```
__init__(param1, param2, param3)
```

Example of docstring on the **init** method.

The **init** method may be documented in either the class level docstring, or as a docstring on the **init** method itself.

Either form is acceptable, but the two should not be mixed. Choose one convention to document the **init** method and be consistent with it.

Note

Do not include the `self` parameter in the `Args` section.

PARAMETER	DESCRIPTION
-----------	-------------

PARAMETER	DESCRIPTION
<code>param1</code>	Description of <code>param1</code> . TYPE: <code>str</code>
<code>param2</code>	obj: <code>int</code> , optional): Description of <code>param2</code> . Multiple lines are supported.
<code>param3</code>	obj: <code>list</code> of obj: <code>str</code>): Description of <code>param3</code> .

__special__

```
__special__()
```

By default special members with docstrings are not included.

Special members are any methods or attributes that start with and end with a double underscore. Any special member with a docstring will be included in the output, if `napoleon_include_special_with_doc` is set to True.

This behavior can be enabled by changing the following setting in Sphinx's conf.py::

```
napoleon_include_special_with_doc = True
```



__special_without_docstring__

```
__special_without_docstring__()
```

example_method

```
example_method(param1, param2)
```

Class methods are similar to regular functions.

 **Note** 

Do not include the `self` parameter in the `Args` section.

PARAMETER	DESCRIPTION
<code>param1</code>	The first parameter.

PARAMETER	DESCRIPTION
<code>param2</code>	The second parameter.

RETURNS	DESCRIPTION
	True if successful, False otherwise.

readonly_property `property`

```
readonly_property()
```

`str`: Properties should be documented in their getter method.

readwrite_property `writable` `property`

```
readwrite_property()
```

`:obj: list` of `:obj: str`: Properties with both a getter and setter should only be documented in their getter method.

If the setter method contains notable behavior, it should be mentioned here.

ExampleError

Bases: `Exception`

Exceptions are documented in the same way as classes.

The **init** method may be documented in either the class level docstring, or as a docstring on the **init** method itself.

Either form is acceptable, but the two should not be mixed. Choose one convention to document the **init** method and be consistent with it.

**Note**

Do not include the `self` parameter in the `Args` section.

PARAMETER	DESCRIPTION
-----------	-------------

PARAMETER	DESCRIPTION
<code>msg</code>	Human readable string describing the exception. TYPE: <code>str</code>
<code>code</code>	obj: <code>int</code> , optional): Error code.

ATTRIBUTE	DESCRIPTION
<code>msg</code>	Human readable string describing the exception. TYPE: <code>str</code>
<code>code</code>	Exception error code. TYPE: <code>int</code>

Attributes

`code` instance-attribute

```
code = code
```

`msg` instance-attribute

```
msg = msg
```

Functions

`__init__`

```
__init__(msg, code)
```

Functions

example_generator

```
example_generator(n)
```

Generators have a `Yields` section instead of a `Returns` section.

PARAMETER	DESCRIPTION
<code>n</code>	The upper limit of the range to generate, from 0 to <code>n</code> - 1. TYPE: <code>int</code>

YIELDS	DESCRIPTION
<code>int</code>	The next number in the range of 0 to <code>n</code> - 1.

Examples:

Examples should be written in doctest format, and should illustrate how to use the function.

```
>>> print([i for i in example_generator(4)])
[0, 1, 2, 3]
```

function_with_pep484_type_annotations

```
function_with_pep484_type_annotations(
    param1: int, param2: str
) -> bool
```

Example function with PEP 484 type annotations.

PARAMETER	DESCRIPTION
<code>param1</code>	The first parameter. TYPE: <code>int</code>
<code>param2</code>	The second parameter. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>bool</code>	The return value. True for success, False otherwise.

function_with_types_in_docstring

```
function_with_types_in_docstring(param1, param2)
```

Example function with types documented in the docstring.

PEP 484_ type annotations are supported. If attribute, parameter, and return types are annotated according to PEP 484_, they do not need to be included in the docstring:

PARAMETER	DESCRIPTION
param1	The first parameter. TYPE: <code>int</code>
param2	The second parameter. TYPE: <code>str</code>

RETURNS	DESCRIPTION
<code>bool</code>	The return value. True for success, False otherwise.

.. _PEP 484: <https://www.python.org/dev/peps/pep-0484/>

module_level_function

```
module_level_function(
    param1, param2=None, *args, **kwargs
)
```

This is an example of a module level function.

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

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. The "(type)" is optional.

    Multiple paragraphs are supported in parameter
    descriptions.
```


PARAMETER	DESCRIPTION
<code>param1</code>	The first parameter. TYPE: <code>int</code>
<code>param2</code>	obj: <code>str</code> , optional): The second parameter. Defaults to None. Second line of description should be indented. DEFAULT: <code>None</code>
<code>*args</code>	Variable length argument list. DEFAULT: <code>()</code>
<code>**kwargs</code>	Arbitrary keyword arguments. DEFAULT: <code>{}</code>

RETURNS	DESCRIPTION
<code>bool</code>	True if successful, False otherwise.
	The return type is optional and may be specified at the beginning of
	the <code>Returns</code> section followed by a colon.
	The <code>Returns</code> section may span multiple lines and paragraphs.
	Following lines should be indented to match the first line.
	The <code>Returns</code> section supports any reStructuredText formatting,
	including literal blocks:: { 'param1': param1, 'param2': param2 }

RAISES	DESCRIPTION
<code>AttributeError</code>	The <code>Raises</code> section is a list of all exceptions that are relevant to the interface.
<code>ValueError</code>	If <code>param2</code> is equal to <code>param1</code> .

Last update: September 13, 2022

Created: September 13, 2022