u2d_msa_sdk 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
`module_level_variable1`	Module level variables may be documented in either the `Attributes` section of the module docstring, or in an inline docstring immediately following the variable. Either form is acceptable, but the two should not be mixed. Choose one convention to document module level variables and be consistent with it. **TYPE:** `int`

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
`attr1`	Description of `attr1`. **TYPE:** `str`
`attr2`	obj:`int`, optional): Description of `attr2`.

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
`param1`	Description of `param1`. **TYPE:** `str`
`param2`	obj: `int`, optional): Description of `param2`. Multiple lines are supported.
`param3`	obj: `list` of :obj: `str`): Description of `param3`.

__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
`param1`	The first parameter.

PARAMETER	DESCRIPTION
`param2`	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 `property` `writable`

```
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
`msg`	Human readable string describing the exception. **TYPE:** `str`
`code`	obj: `int`, optional): Error code.

ATTRIBUTE	DESCRIPTION
`msg`	Human readable string describing the exception. **TYPE:** `str`
`code`	Exception error code. **TYPE:** `int`

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
`n`	The upper limit of the range to generate, from 0 to `n` - 1. **TYPE:** `int`

YIELDS	DESCRIPTION
`int`	The next number in the range of 0 to `n` - 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
`param1`	The first parameter. **TYPE:** `int`
`param2`	The second parameter. **TYPE:** `str`

RETURNS	DESCRIPTION
`bool`	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:** `int`
`param2`	The second parameter. **TYPE:** `str`

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

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

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

Last update: September 13, 2022

Created: September 13, 2022