# docs-example

***Release 0.1.0***

**docs-example contributors**

**Jun 23, 2022**

# GET STARTED

CHAPTER

ONE

# INTRODUCTION

CHAPTER

TWO

# INSTALLATION

## 2.1 Prerequisites

## 2.2 Installation

## 2.3 Verification

CHAPTER

# THREE

# HOW TO SAY HELLO

```
>>> from docs_example.example1 import say_hello
>>> say_hello()
hello
```

CHAPTER

# FOUR

# HOW TO SAY HI

```
>>> from docs_example.example1 import say_hi
>>> say_hi()
hi
```

CHAPTER

FIVE

# SPHINX EXTENSION

## 5.1 mermaid

| https://sphinxcontrib-mermaid-demo.readthedocs.io/en/latest/

CHAPTER

SIX

# ENGLISH

CHAPTER

# SEVEN

CHAPTER

# EIGHT

# DOCS_EXAMPLE.EXAMPLE1

docs_example.example1.**say_hello**(*n: int = 1*)
Print hello n times to terminal.

**Parameters** **n** (*int*) – Print hello n times to terminal.

CHAPTER

NINE

# DOCS_EXAMPLE.EXAMPLE2

docs_example.example2.**say_hi**(*n: int = 1*)
Print hi n times to terminal.

**Parameters** **n** (*int*) – Print hi n times to terminal.

CHAPTER

# TEN

# DOCS_EXAMPLE.STYLE_GUIDE

| | |
|---|---|
| *ExampleClass* | The summary line for a class docstring should fit on one line. |
| *example_generator* | Generators have a `Yields` section instead of a `Returns` section. |
| *module_level_function* | This is an example of a module level function. |

## 10.1 ExampleClass

**class** docs_example.style_guide.**ExampleClass**(*arg1: int*, *arg2: Optional[str] = None*, *arg3: str = 'item1'*, *arg4: Optional[dict] = None*)

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

Describe the function of the class in a few sentences, including but not limited to what the class is and what it can do.

In the docstring, we can list some items with the following format:

- item1
- item2
- item3
- item4

If we want to quote a URL, then we can quote it like OpenMMLab. Note that there needs to be a space between the title and the link, otherwise it will not render successfully. We can also quota a link like OpenMMLab.

**Note:** If the class has something important to remind the user, we can show it in the note block.

**Warning:** If the class has something important to warn the user, such as a change in the interface, we can show it in the warning block.

Sometimes we want to use formulas to represent some definitions, we can write them within lines like $e^{i\pi}+1=0$. Or we can write it between lines like.

$$e^{i\pi}+1=0$$

**Note:** Of course, we can also write the formula in block like $e^{i\pi} + 1 = 0$ or the following format.

$$e^{i\pi} + 1 = 0$$

To make it easier for users to get started quickly, we can give some introductory examples in the example block.

### Examples

```
>>> # initialization
>>> obj = ClassDocstring(1)
>>> # give a few more examples
>>> obj = ClassDocstring(1, 'second parameter')
```

**Parameters**

- **arg1** (*int*) – `arg1` is the first parameter. If we want to quote a link to explain the argument, we can say that see more details at website.
- **arg2** (*str, optional*) – `arg2` is the second parameter. Defaults to None. Note that if the default value is None, then you need to add optional to the parameter type. If the description of a parameter is too long, just indent it with a new line.
- **arg3** (*str*) – Of course, We can list some optional values with the following format. Defaults to `item1`.
  - item1
  - item2. If the description of a parameter is too long, just indent it with a new line. Be careful to the indentation alignment between lines otherwise there will be unexpected rendering result.
- **arg4** (*dict, optional*) – If the parameter is a dictionary, we can describe it with the following format. Defaults to None.
  - key1: This is a shot description of the item.
  - key1: This is a shot description of the item.

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

**Note:** Note that the usage of `double backticks`, *single backticks*, and "double quotation marks" are different. In reStructured syntax, `double backticks` means a piece of code. *single backticks* means italics. "double quotation marks" has no special meaning, but can be used to represent a string. The usage of *single backticks* is different from that in Markdown, so you should pay attention to it.

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). Note that the `Attributes` section is optional.

**args1**
Description of *args1*.

**Type** int

**args2**
Description of *args2*.

**Type** str, optional

**arg3**
Doc comment *inline* with attribute

**arg4**
Doc comment *before* attribute, with type specified

**Type** dict or None

**argument_list_changed**(*arg1: int*, *arg2: str = ''*) → None
If the argument list of a method have changed, we need to reflect that in the docstring.

---

**Note:** More specifically, if the parameter is newly added, we can use the *New in version 1.1.1.* which *1.1.1* is the version parameter was added. If the meaning of the parameter are changed, we can use the *Changed in version 1.1.1.*.

---

**Warning:** If the parameter was renamed, we need to tell user the interface was changed in the warning block.

**Parameters**

- **arg1** (*int*) – arg1 is the first parameter.
- **arg2** (*str*) – arg2 is the second parameter. Defaults to ''. *New in version 1.1.1.*

**property class_name**
Properties should be documented in their getter method.

**Type** str

**property owner**
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.

**Type** str

**return_dict**(*arg1: int*, *arg2: Union[str, list]*, *arg3: dict*) → dict
Summarize the function of the method in one sentence.

Describe the function of the class in a few sentences. Of course, it is not required.

**Parameters**

- **arg1** (*int*) – arg1 is the first parameter.
- **arg2** (*str | list*) – arg2 is the second parameter. This parameter may be of type string or list.
- **arg3** (*dict*) – We can describe the parameter with the following format.
  - key1: This is a shot description of the item.
  - key1: This is a shot description of the item.

**Returns** Return the output where the keys denote A meaning and values denote B. Be careful to the indentation alignment between lines otherwise there will be unexpected rendering result.

**Return type** dict

**return_string**(*arg1: int*, *arg2: Union[str, list]*, *arg3: dict*) → str

Summarize the function of the method in one sentence.

Describe the function of the class in a few sentences. Of course, it is not required.

---

**Note:** Do not include the *self* parameter in the `Args` section.

---

**Parameters**

- **arg1** (*int*) – `arg1` is the first parameter.
- **arg2** (*str | list*) – `arg2` is the second parameter. This parameter may be of type string or list.
- **arg3** (*dict*) – We can describe the parameter with the following format.
  - key1: This is a shot description of the item.
  - key1: This is a shot description of the item.

**Returns** Return the output.

**Return type** str

**return_tuple**() → tuple

Return a tuple.

**Returns** Returns a tuple (a, b), where a is xxx, and b is xxx.

**Return type** tuple

## 10.2 example_generator

**class** docs_example.style_guide.**example_generator**(*n*)

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

**Parameters** **n** (*int*) – The upper limit of the range to generate, from 0 to *n* - 1.

**Yields** *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]
```

## 10.3 module_level_function

class docs_example.style_guide.**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.
```

**Parameters**

- **param1** (*int*) – The first parameter.
- **param2** (str, optional) – The second parameter. Defaults to None. Second line of description should be indented.
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

**Returns**

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
}
```

**Return type** bool

**Raises**

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

CHAPTER

ELEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

# INDEX