Basic Sphinx Demo Project

Ipora Possantti

Mar 06, 2023

CONTENTS

A cool image

A Figure

Fig. 1: This is the caption of the figure (a simple paragraph).

Danger: Beware killer rabbits!

This example shows a basic Sphinx project with Read the Docs. You're encouraged to view it to get inspiration and copy & paste from the files in the source code. If you are using Read the Docs for the first time, have a look at the official Read the Docs Tutorial.

Table 1: Doctable

Field 1	Field 2	Field 3
C1	1.3	OK
c12	0.2	Map
c13	100.2	Table

Table 2: Table Title

Heading row 1, column 1	Heading row 1, column 2	Heading row 1, column 3
Row 1, column 1		Row 1, column 3
Row 2, column 1	Row 2, column 2	Row 2, column 3

docs/
: A basic Sphinx project lives in `docs/`. All the `*.rst` make up sections in the documentation.

.readthedocs.yaml
: Read the Docs Build configuration is stored in `.readthedocs.yaml`.

docs/conf.py
: Both the configuration and the folder layout follow Sphinx default conventions. You can change the Sphinx configuration values in this file

docs/requirements.txt and docs/requirements.in
: Python dependencies are pinned (uses pip-tools). Make sure to add your Python dependencies to `requirements.txt` or if you choose [pip-tools](https://pip-tools.readthedocs.io/en/latest/), edit `docs/requirements.in` and remember to run `pip-compile docs/requirements.in`.

docs/api.rst
: By adding our example Python module `lumache` in the reStructuredText directive `:autosummary:`, Sphinx will automatically scan this module and generate API docs.

docs/usage.rst
: Sphinx can automatically extract API documentation directly from Python modules, using for instance the `:autofunction:` directive.

lumache.py
: API docs are generated for this example Python module - they use *docstrings* directly in the documentation, notice how this shows up in the rendered documentation.

Git tags versioning
: We use a basic versioning mechanism by adding a git tag for every release of the example project. All releases and their version numbers are visible on example-sphinx-basic.readthedocs.io.

README.rst
: Contents of this `README.rst` are visible on Github and included on the documentation index page (Don't Repeat Yourself).

Questions / comments
: If you have questions related to this example, feel free to can ask them as a Github issue here.

CHAPTER

ONE

EXAMPLE PROJECT USAGE

This project has a standard Sphinx layout which is built by Read the Docs almost the same way that you would build it locally (on your own laptop!).

You can build and view this documentation project locally - we recommend that you activate a local Python virtual environment first:

```
# Install required Python dependencies (Sphinx etc.)
pip install -r docs/requirements.txt

# Enter the Sphinx project
cd docs/

# Run the raw sphinx-build command
sphinx-build -M html . _build/
```

You can also build the documentation locally with `make`:

```
# Enter the Sphinx project
cd docs/

# Build with make
make html

# Open with your preferred browser, pointing it to the documentation index page
firefox _build/html/index.html
```

CHAPTER

TWO

USING THE EXAMPLE IN YOUR OWN PROJECT

If you are new to Read the Docs, you may want to refer to the Read the Docs User documentation.

If you are copying this code in order to get started with your documentation, you need to:

1. place your `docs/` folder alongside your Python project. If you are starting a new project, you can adapt the *pyproject.toml* example configuration.
2. use your existing project repository or create a new repository on Github, GitLab, Bitbucket or another host supported by Read the Docs
3. copy `.readthedocs.yaml` and the `docs/` folder into your project.
4. customize all the files, replacing example contents.
5. add your own Python project, replacing the `pyproject.toml` configuration and `lumache.py` module.
6. rebuild the documenation locally to see that it works.
7. *finally*, register your project on Read the Docs, see Importing Your Documentation.

CHAPTER

THREE

READ THE DOCS TUTORIAL

To get started with Read the Docs, you may also refer to the Read the Docs tutorial. It provides a full walk-through of building an example project similar to the one in this repository.

CHAPTER

FOUR

CROSS-REFS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer eu egestas ipsum. Curabitur aliquam, nulla eget ornare commodo, nisl lectus auctor felis, quis facilisis libero justo ac nisl. Maecenas efficitur arcu sem, vitae vehicula purus posuere vitae. Donec at justo justo. Phasellus eros nisl, malesuada quis convallis eu, gravida vel magna. Mauris varius nunc vel dui fringilla pellentesque. Phasellus eget laoreet ligula. Mauris sed aliquam dui, ac lacinia nisi.

CHAPTER

FIVE

WELCOME TO LUMACHE'S DOCUMENTATION!

Lumache (/lu'make/) is a Python library for cooks and food lovers that creates recipes mixing random ingredients. It pulls data from the Open Food Facts database and offers a *simple* and *intuitive* API.

Check out the *Usage* section for further information, including how to *Installation* the project.

Note: This project is under active development.

5.1 Contents

5.1.1 Usage

Installation

To use Lumache, first install it using pip:

```
(.venv) $ pip install lumache
```

Creating recipes

To retrieve a list of random ingredients, you can use the `lumache.get_random_ingredients()` function:

The `kind` parameter should be either `"meat"`, `"fish"`, or `"veggies"`. Otherwise, `lumache.get_random_ingredients()` will raise an exception.

For example:

```
>>> import lumache
>>> lumache.get_random_ingredients()
['shells', 'gorgonzola', 'parsley']
```

Referencing things in other pages

Lorem ipsum dolor sit amet *some disadvantages*:, consectetur adipiscing elit. Integer eu egestas ipsum. Curabitur aliquam, nulla eget ornare commodo, nisl lectus auctor felis, quis facilisis libero justo ac nisl. As illustrated in Fig. 1 and more.

5.1.2 I/O files

text.

5.1.3 API Reference

The following section outlines the API of disnake.

Note: This module uses the Python logging module to log diagnostic and errors in an output independent way.

Modules

thelib	Routines for working with triangles.

thelib

Routines for working with triangles.

The two modules inside of this package are packed with useful features for the programmer who needs to support triangles:

lumache
: This module provides a full-fledged *Triangle* object that can be instantiated and then asked to provide all sorts of information about its properties.

other
: For the programmer in a hurry, this module offers quick functions that take as arguments the three side lengths of a triangle, and perform a quick computation without the programmer having to make the extra step of creating an object.

thelib.lumache	Lumache - Python library for cooks and food lovers.

thelib.lumache

Lumache - Python library for cooks and food lovers.

This is a Python docstring, we can use reStructuredText syntax here!

```
# Import lumache
import lumache
```

(continues on next page)

(continued from previous page)

```
# Call its only function
get_random_ingredients(kind=["cheeses"])
```

Functions

get_random_ingredients([kind])	Return a list of random ingredients as strings.

Classes

MyClass([s_name])	A new Object

Exceptions

InvalidKindError	Raised if the kind is invalid.

thelib.other	other - Python library for cooks and food lovers.

thelib.other

other - Python library for cooks and food lovers.

This is a Python docstring, we can use reStructuredText syntax here!

```
# Import lumache
import lumache

# Call its only function
get_random_ingredients(kind=["cheeses"])
```

Functions

get_ordinal_ingredients([kind])	Return a list of obvios ingredients as strings.

Exceptions

InvalidKindError	Raised if the kind is invalid.

Lumache functions

thelib.lumache.**get_random_ingredients**(*kind=None*)

Return a list of random ingredients as strings.

Parameters
kind (*list[str] or None*) – Optional "kind" of ingredients.

Raises
lumache.InvalidKindError – If the kind is invalid.

Returns
The ingredients list.

Return type
list[str]

Lumache classes

class thelib.lumache.**MyClass**(*s_name='MyName'*)

A new Object

do_stuff(*s_str1*, *n_value*)

A demo method.

Parameters

- **s_str1** (*str*) – string to print.
- **n_value** (*float*) – value to print.

Returns
a concatenated string

Return type
str

PYTHON MODULE INDEX

t

INDEX