

Collaborative Software Design

Documentation Basics

Manuel Giffels (Manuel.Giffels@kit.edu)
KSETA Course — October 2025



Why is Documentation Essential?

- Facilitates collaboration in multi-developer projects
- Supports reproducibility
- Reduces onboarding time for new contributors
- Enables long-term maintenance (often beyond your PhD)

Good code is only useful if people can understand and reuse it.



Types of Documentation

Type	Purpose	Target Audience
Readme	Overview, quick start, examples	User, developers
API Documentation	Usage of functions/classes	Developers
Architecture Documentation	System design, design decisions	Developers
Inline Comments	Clarify tricky code	Developers
User Guide/Tutorials	How-to instructions	End users
Developer Guides	Helps onboarding	New Developers

Reasonable Structure of the Readme

- Project name and brief description
- Installation/setup instructions
- Quick start example
- Dependencies and requirements
- License and citation information
- Contribution guidelines
- Contact details

Think of the README as the „front door“ to your project

README.md2.49 KiB

CodePreview

Scientific Calculator Project

This is a Python-based scientific calculator implemented using Tkinter. The project is structured so that arithmetic operations (+, -, *, /) work out of the box, while scientific functions like `sin`, `cos`, `tan`, `exp`, `log`, etc., are intended to be implemented by students in a separate module.

Project Structure

```
calculator_project/
├── calculator/           # Core calculator code
│   ├── __init__.py
│   ├── core.py           # Basic arithmetic functions
│   ├── functions.py      # Placeholder for scientific functions (students implement)
│   ├── constants.py      # Placeholder for scientific constants (students implement)
│   └── gui.py            # Tkinter GUI
├── tests/               # Unit tests
│   ├── __init__.py
│   └── calculator_t/
│       ├── __init__.py
│       ├── test_functions.py # Students add unit tests for scientific functions here
│       ├── test_constants.py # Students add unit tests for scientific constants here
│       ├── test_calculator.py
│       └── test_gui.py
├── run_test.sh          # Script to run all tests
├── run_coverage.sh      # Script to run coverage
├── setup.py
└── README.md
```

Setup Instructions

1. Create a virtual environment:

```
python -m venv venv
source venv/bin/activate # Linux/macOS
venv\Scripts\activate   # Windows
```

Inline Code Documentation

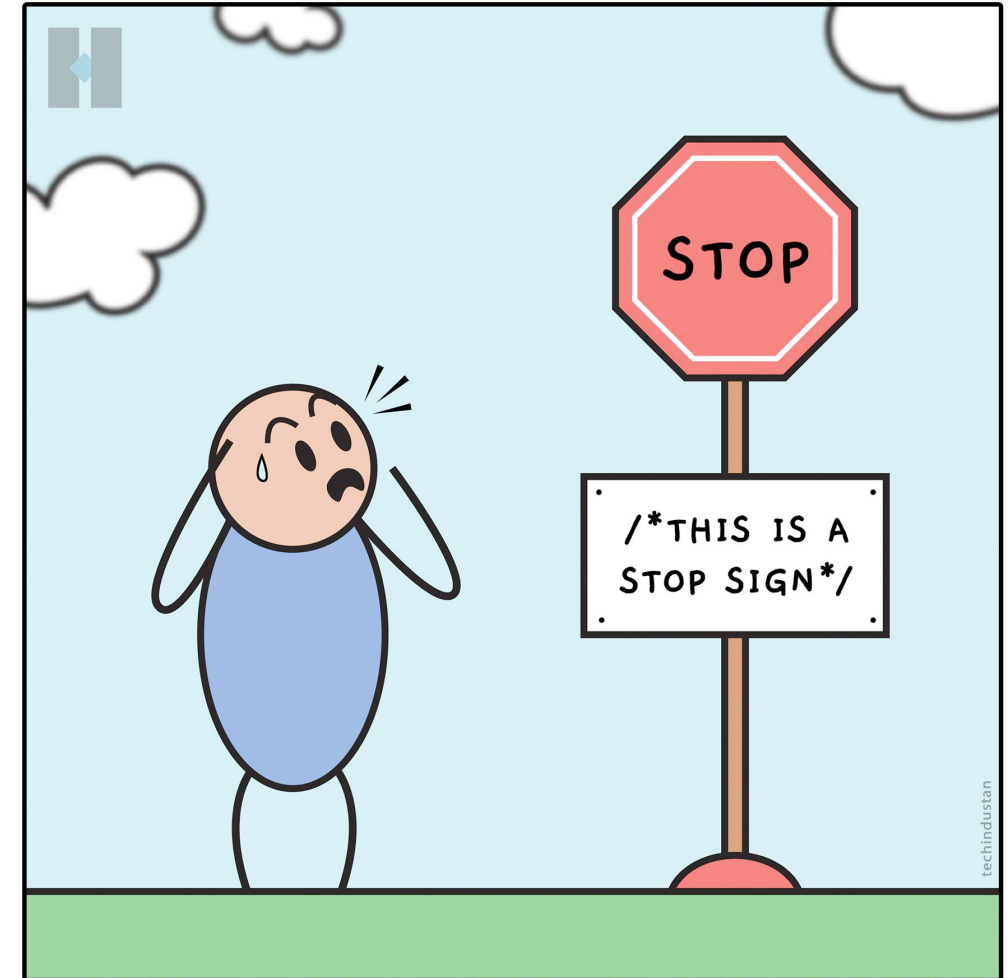
- Comment why, not what the code does

```
@property
def _async_lock(self):
    # Create lock once tardis event loop is running.
    # To avoid got Future <Future pending> attached to a different loop exception
    if not self._lock:
        self._lock = asyncio.Lock()
    return self._lock
```

- Keep comments close to the relevant code
- Avoid outdated comments — update as code evolves
- Use docstrings (Python) for functions and classes

```
@property
def configuration(self) -> AttributeDict:
    """
    Property to provide access to SiteAdapter specific configuration.
    :return: returns the Site Adapter specific configuration
    :rtype: AttributeDict
    """
    return getattr(Configuration(), self.site_name)
```

BEGINNER'S COMMENTING ON CODE LIKE



www.techindustan.com - Finest IT Services Company

f t i /techindustan

API Documentation Tools

- Use tools for automatic generation
 - Python → [Sphinx](#)
 - C++ → [Doxygen](#)
 - Java → [JavaDoc](#)
 - JS/TS → [TypeDoc](#)
- Integrate with CI to keep docs updated
- Encourage clear, consistent docstring formats (e.g. Sphinx format)
- Utilization of Python Type Hints helps — no need to specify types in the docstring as well

```
class ResourceStatus(Enum):  
    """  
    Status of the resource at the resource provider (batch system, cloud provider, etc.)  
    """  
  
    Booting = 1  
    Running = 2  
    Stopped = 3  
    Deleted = 4  
    Error = 5  
  
class SiteAdapter(metaclass=ABCMeta):  
    """  
    Abstract base class defining the interface for SiteAdapters which provide  
    access to various Cloud APIs and batch systems in order to manage  
    opportunistic resources.  
    """  
  
    @property  
    def configuration(self) -> AttributeDict:  
        """  
        Property to provide access to SiteAdapter specific configuration.  
        :return: returns the Site Adapter specific configuration  
        :rtype: AttributeDict  
        """  
        return getattr(Configuration(), self.site_name)  
  
    @abstractmethod  
    async def deploy_resource(  
        self, resource_attributes: AttributeDict  
    ) -> AttributeDict:  
        """  
        Abstract method to define the interface to deploy a new resource at a  
        resource provider.  
        :param resource_attributes: Contains describing attributes of the resource,  
        defined in the :py:class:`~tardis.resources.drone.Drone` implementation!  
        :type resource_attributes: AttributeDict  
        :return: Contains updated describing attributes of the resource.  
        :rtype: AttributeDict  
        """  
        raise NotImplementedError
```

API Documentation Tools

- Use tools for automatic generation
 - Python → [Sphinx](#)
 - C++ → [Doxygen](#)
 - Java → [JavaDoc](#)
 - JS/TS → [TypeDoc](#)
- Integrate with CI to keep docs updated
- Encourage clear, consistent docstring formats (e.g. Sphinx format)
- Utilization of Python Type Hints helps — no need to specify types in the docstring as well

🏠 / tardis / tardis package / tardis.interfaces package / tardis.interfaces.siteadapter module [View page source](#)

tardis.interfaces.siteadapter module

```
class tardis.interfaces.siteadapter.ResourceStatus(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None) \[source\]
```

Bases: `Enum`

Status of the resource at the resource provider (batch system, cloud provider, etc.)

Booting= 1

Deleted= 4

Error= 5

Running= 2

Stopped= 3

```
class tardis.interfaces.siteadapter.SiteAdapter \[source\]
```

Bases: `object`

Abstract base class defining the interface for SiteAdapters which provide access to various Cloud APIs and batch systems in order to manage opportunistic resources.

property configuration: `AttributeDict`

Property to provide access to SiteAdapter specific configuration. :return: returns the Site Adapter specific configuration :rtype: AttributeDict

```
abstract async def deploy_resource(resource_attributes: AttributeDict) → AttributeDict \[source\]
```

Abstract method to define the interface to deploy a new resource at a resource provider. :param resource_attributes: Contains describing attributes of the resource, defined in the `Drone` implementation! :type resource_attributes: AttributeDict :return: Contains updated describing attributes of the resource. :rtype: AttributeDict

Setting Up Sphinx Documentation for a Python Project

- Install Sphinx package
`pip install sphinx`
- Create docs directory in scientific calculator project
`mkdir docs && cd docs`
- Run sphinx-quickstart and fill the information
`sphinx-quickstart`
- Install additional sphinx packages
`pip install sphinx_rtd_theme sphinx-autodoc-typehints`

Setting Up Sphinx Documentation for a Python Project

- Open `conf.py` and update the configuration

```
# Configuration file for the Sphinx documentation builder.
#
# For the full list of built-in configuration values, see the documentation:
# https://www.sphinx-doc.org/en/master/usage/configuration.html

# -- Project information -----
# https://www.sphinx-doc.org/en/master/usage/configuration.html#project-information

project = 'Scientific Calculator'
copyright = '2025, Manuel Giffels'
author = 'Manuel Giffels'
release = '0.1.0'

# -- General configuration -----
# https://www.sphinx-doc.org/en/master/usage/configuration.html#general-configuration

extensions = [
    'sphinx.ext.autodoc',
    'sphinx.ext.napoleon', # supports Google & NumPy style docstrings
    'sphinx_autodoc_typehints'
]

templates_path = ['_templates']
exclude_patterns = ['_build', 'Thumbs.db', '.DS_Store']

# -- Options for HTML output -----
# https://www.sphinx-doc.org/en/master/usage/configuration.html#options-for-html-output

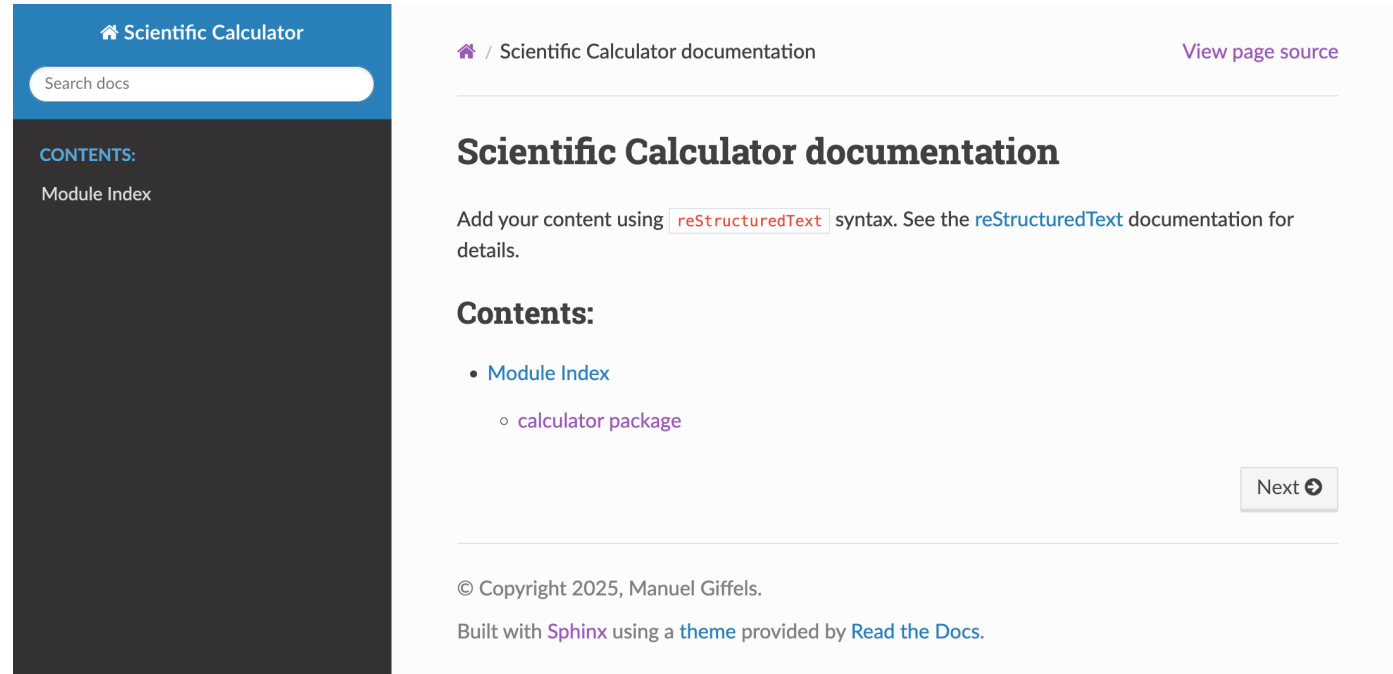
html_theme = 'sphinx_rtd_theme'
html_static_path = ['_static']
```

Generate Sphinx Documentation for a Python Project

- Generate the API documentation
`sphinx-apidoc -o source/api ../calculator`
- Add Module Index to the table of content (index.rst)

```
❏. toctree::  
    :maxdepth: 2  
    :caption: Contents:  
  
    Module Index <source/api/modules>
```

- Create the html output
`make html`
- Check output in a web browser
`open _build/html/index.html`



The screenshot displays the generated Sphinx documentation for a project named 'Scientific Calculator'. The interface is split into two main sections. On the left, a dark-themed sidebar contains a search bar and a 'CONTENTS:' section listing 'Module Index'. The main content area on the right has a light background and includes a breadcrumb trail, a 'View page source' link, the title 'Scientific Calculator documentation', a paragraph about reStructuredText syntax, a 'Contents:' section with a bulleted list of 'Module Index' (which further links to 'calculator package'), and a 'Next' button with a right arrow. The footer contains copyright information for 2025 and mentions the use of Sphinx and Read the Docs theme.

Scientific Calculator

Search docs

CONTENTS:

Module Index

Scientific Calculator documentation

View page source

Add your content using `reStructuredText` syntax. See the [reStructuredText](#) documentation for details.

Contents:

- [Module Index](#)
 - [calculator package](#)

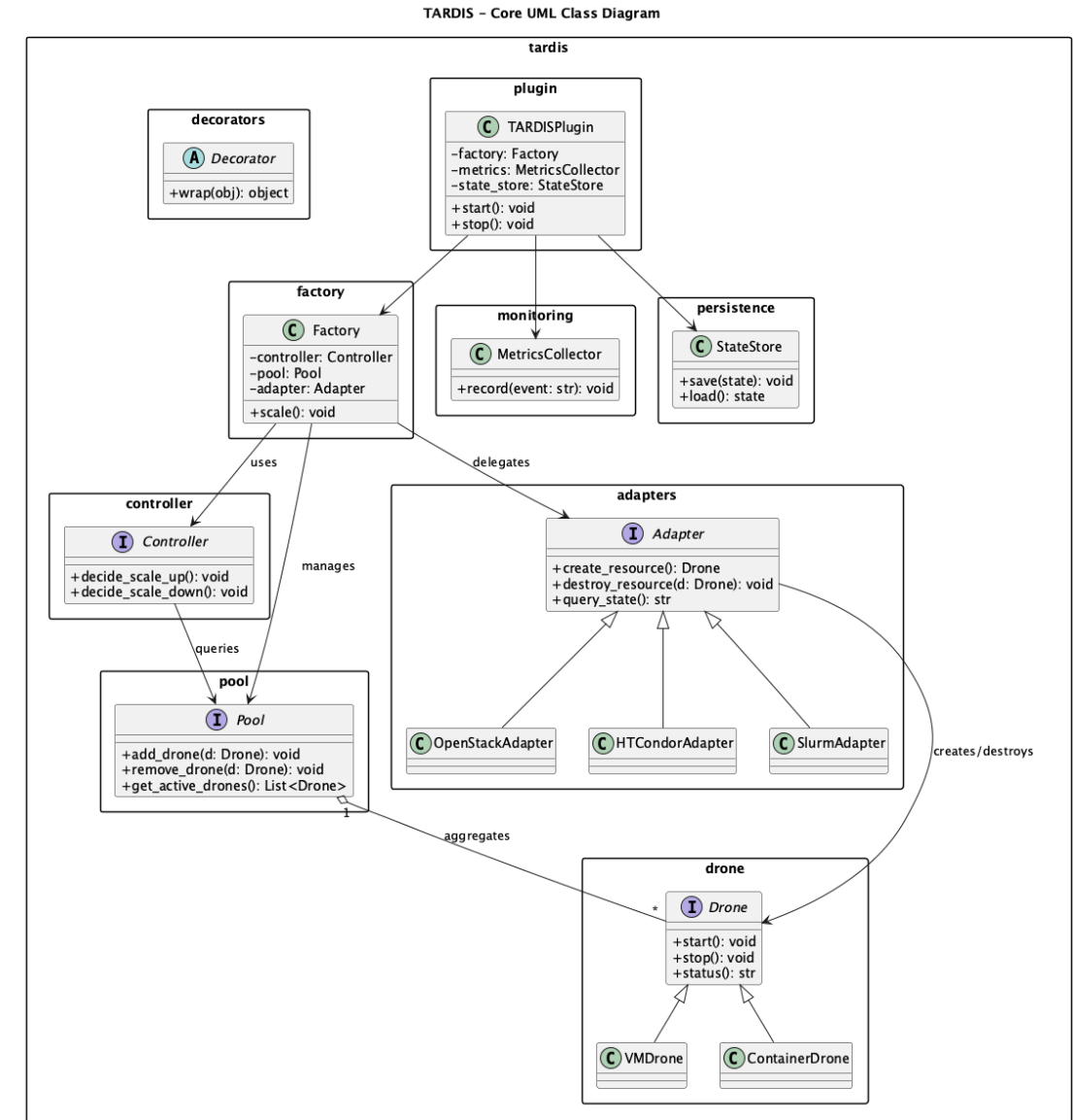
Next ➔

© Copyright 2025, Manuel Giffels.

Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

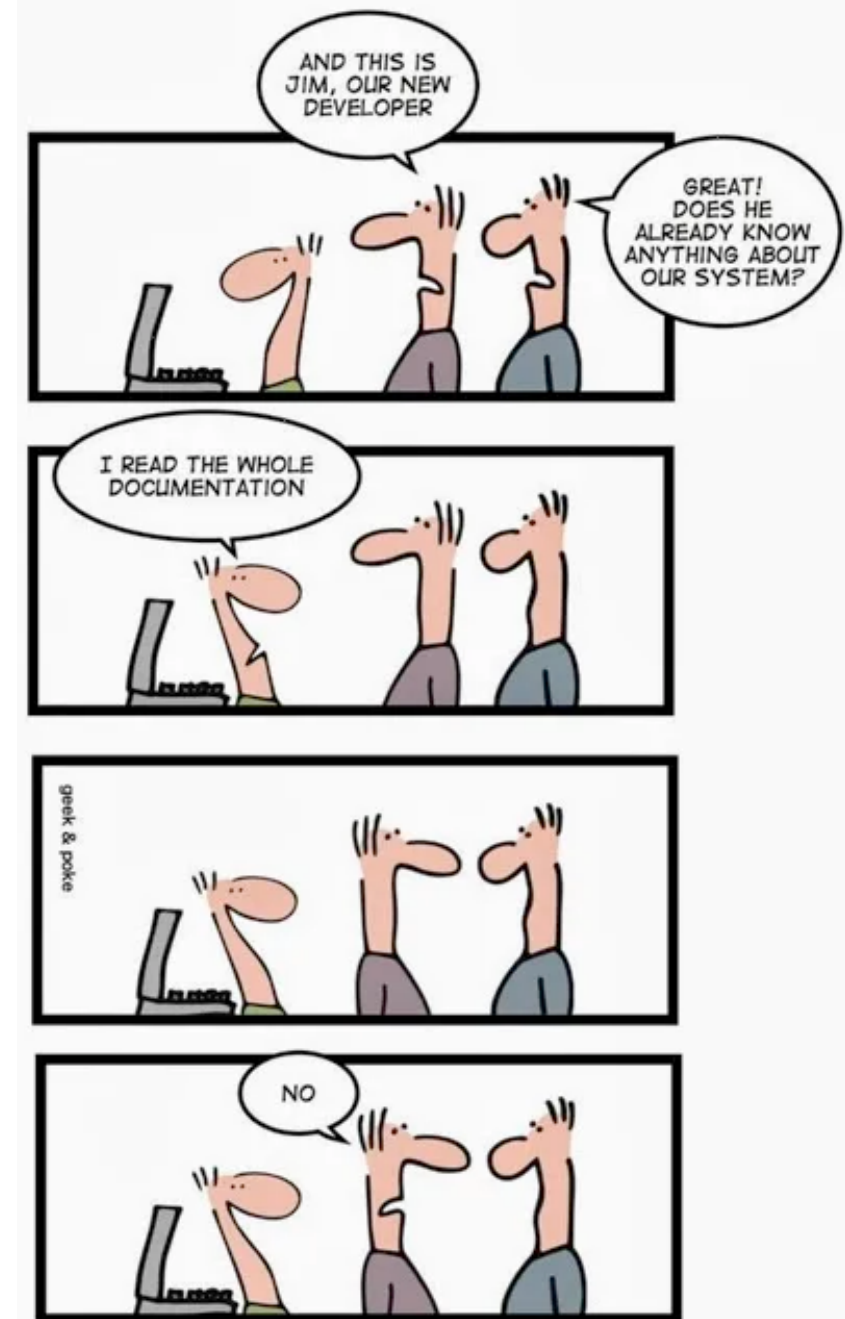
Architectural & Design Documentation

- Describe the overall structure, key components, dependencies, basic principles, etc.
- Tools: Markdown, Unified Modelling Language (UML), also generators available
- Keep diagrams simple & version controlled
- Document also major design decisions, consequences and the reason for it — called ADR (Architecture Decision Record)
- Usually very reasonable on large projects with many and changing contributors are involved



Documentation Best Practices

- Treat documentation like code
 - Use version control (Git)
 - Use CI to build it
 - Consider it in merge requests and the corresponding review process
 - Assign documentation tasks explicitly
- Encourage everyone to contribute to the documentation — it is not just one person's job
- Link documentation updates to feature changes in issues and merge requests



Common Pitfalls

- Assuming code is self-documenting
→ even well-written code rarely conveys design decisions, etc.
- Underestimating the lifetime of code
→ Academic code often outlives the authors
- Documentation of a project left to the end
→ outdated quickly, not fun writing entire documentation at once
- Overly detailed or too sparse documentation
→ hard to maintain or simply useless
- Unclear audience targeting
→ wrong level of detail, no benefits
- Only auto-generated docs
→ Lack of context

```
// the main entry method for the program, which
// the java command line program will execute.
public static void main(String[] args) {
    ..// prompt the user for their name using System.out, which
    ..// is a PrintStream class. The PrintStream class has a
    ..// method called println, which will output the text
    ..// passed to the console (so that the user can see it).
    ..// and then print a newline.
    ..System.out.println("Welcome to my program! What is your name? ");
    ..
    ..// this makes a new scanner, which can read from
    ..// STDIN, located at System.in. The scanner lets us look
    ..// for tokens, aka stuff the user has entered.
    ..Scanner sc = new Scanner(System.in);
    ..
    ..// this will create a new s
    ..// and set the value to wha
    ..String name = sc.nextLine()
    ..
    ..// finally we use println
```

**AND POINTLESS OR POORLY
WRITTEN DOCUMENTATION CAN
BE WORSE THAN NONE AT ALL.**

BY ALEC MCEACHRAN
© KPV 2021

Take Away Messages

- Documentation is part of the project from the very beginning
- Keep it clear, concise and evolving
- Document the „why“, and „how“
- Add examples and a minimal runnable demo
- Use tools & workflows to reduce overhead
- Use consistent vocabulary and structure
- Good documentation means smoother collaboration and reproducibility



BY ALEC MCEACHRAN
© KPV 2021



Questions?



Hands-on

Hands-on

- Again work together in teams of two people
- Set-up a simple API documentation using Sphinx
- Add the build of the API documentation to the GitLab CI/CD