***Docstring Documentation***

We are using tools like Sphinx and MkDocs to generate documentation from Python docstrings. Sphinx offers advanced customization options, ideal for creating detailed technical documentation, while MkDocs provides a simpler, static site generator for easy deployment. Together, these tools automate the process of turning in-code docstrings into well-structured, user-friendly documentation.

***Sphinx:***

Sphinx is a powerful documentation generator that converts reStructuredText files into various formats like HTML, PDF, and ePub. It's widely used for Python projects to create well-structured, readable documentation with features like cross-referencing, automatic code documentation, and theming.

**Step 1: Write Python Code with Docstrings**

First, create a simple Python project and add docstrings to document the code.

**File Structure:**

**```plaintext**

**my\_project/**

**│**

**├── docs/**

**│ └── conf.py (Sphinx config, auto-generated)**

**│ └── src/**

**│ └── my\_module.py**

**├── requirements.txt (optional: for dependencies)**

**└── README.md**

**```**

**File: `docs/src/my\_module.py`**

**```python**

**class Calculator:**

**"""**

**A simple calculator class to perform basic operations.**

**Methods**

**-------**

**add(a, b)**

**Adds two numbers.**

**subtract(a, b)**

**Subtracts second number from first number.**

**multiply(a, b)**

**Multiplies two numbers.**

**divide(a, b)**

**Divides first number by second number. Raises ZeroDivisionError for division by zero.**

**"""**

**def add(self, a, b):**

**"""Returns the sum of a and b."""**

**return a + b**

**def subtract(self, a, b):**

**"""Returns the difference when b is subtracted from a."""**

**return a - b**

**def multiply(self, a, b):**

**"""Returns the product of a and b."""**

**return a \* b**

**def divide(self, a, b):**

**"""**

**Returns the quotient of a divided by b.**

**Raises**

**------**

**ZeroDivisionError**

**If b is 0.**

**"""**

**if b == 0:**

**raise ZeroDivisionError("Cannot divide by zero!")**

**return a / b**

**```**

**Step 2: Set Up Sphinx for Documentation**

1. **Install Sphinx**

**```bash**

**pip install sphinx**

**```**

1. **Initialize Sphinx Documentation**

Navigate to the project folder:

**```bash**

**cd my\_project**

**```**

1. **Initialize Sphinx in the `docs` folder:**

**```bash**

**sphinx-quickstart docs**

**```**

This will generate some default files in `docs/`. Answer the prompts to set up the Sphinx project. Set `source` folder and `build` folder names as the prefer.

1. **Edit `docs/conf.py`**

Ensure that the source path of the code is added to the system path by adding this to `conf.py`:

**```python**

**import os**

**import sys**

**sys.path.insert(0, os.path.abspath('../src'))**

**extensions = ['sphinx.ext.autodoc']**

**```**

1. **Edit `docs/source/index.rst`**

Ensure that the source module is added to this file:

**```rst**

**.. automodule:: my\_module**

**:members:**

**```**

1. **Build HTML Documentation**

To build the HTML documentation, run the following command:

**```bash**

**make html**

**```**

This will generate HTML documentation in the `docs/build/html` folder. Open `index.html` from this folder to view the documentation.

**Alternative Way:**

**4. sphinx-apidoc -o source/ src/**

After this go to ‘index.rst’ and include:

**```**

**.. toctree::**

**:maxdepth: 2**

**:caption: Contents:**

**modules**

**```**

***MkDocs:***

MkDocs is a static site generator specifically designed for creating project documentation. It uses simple Markdown files to write content and can automatically convert them into a full website. MkDocs is widely used for generating documentation for software projects due to its ease of use, flexibility, and support for various themes.

To automatically generate documentation using `mkdocs` and `pdoc` for the Python code (`my\_module.py`), follow these steps:

1. **Install the required tools:**

Make sure we have `mkdocs`, `pdoc`, and `mkdocs-material` installed:

```bash

pip install mkdocs pdoc mkdocs-material

```

1. **Set up the `mkdocs` structure:**

Create a directory for the project:

**```bash**

**mkdir my\_project**

**cd my\_project**

**```**

1. **Initialize `mkdocs` in the project:**

**```bash**

**mkdocs new .**

**```**

This will create an initial `mkdocs.yml` configuration file and a `docs` folder.

1. **Generate documentation using `pdoc`:**

Run `pdoc` to generate Markdown files for the Python code:

**```bash**

**pdoc --output-dir docs src/my\_module.py**

**```**

This will generate `my\_module.md` in the `docs` folder, which contains the auto-generated documentation for the code.

1. **Update `mkdocs.yml`:**

Open the `mkdocs.yml` file and edit it to include the generated `my\_module` documentation.

Here's an example of how we might modify it:

**```yaml**

**site\_name: My Project Documentation**

**theme:**

**name: material**

**nav:**

**- Home: index.md**

**- My Module: my\_module.html**

**```**

1. **Serve the documentation:**

We can now build and serve the documentation using `mkdocs`:

**```bash**

**mkdocs serve**

**```**

Visit `**http://127.0.0.1:8000**` in the browser to see the generated documentation for the `my\_module.py` file.

1. **Site Building documentation:**

To Build the site and store in a site folder use:

**```bash**

**mkdocs build**

**mkdocs gh-deploy**

**```**

Mkdocs build command build the site and store it in site folder.

Now the documentation will automatically be generated from the code in `my\_module.py` using `pdoc` and served using `mkdocs`.