Continuous Documentation for Your Python code

by Anastasiia Tymoshchuk
Associate Director of Engineering @ Soundwide
Few words about myself

- Associate Director of Engineering at Soundwide in Berlin
- PyBerlin organiser
  https://www.meetup.com/PyBerlin/
- 11 years in software development
- 7 years in Python
- Happy Pythonista 🐍😊
Do you document your code?
Me 10 years ago..

DOCUMENTATION?
WHERE WE'RE GOING, WE DON'T NEED DOCUMENTATION
Let the future begin...
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/why/{param_1}" )
async def why_function(param_1: int, some: Optional[str] = None):
    return {"message": "What am I doing here?", "param_1": param_1, "some": some}
Sad Code

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {"message": "What am I doing here?", "param_1": param_1, "some": some}
```
```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {'message': 'Hello World!'}

@app.get("/why/{param_1}" Notícias)
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```
```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/why/{param_1}"/
async def why_function(param_1: int, some: Optional[str] = None):
    return {"message": "What am I doing here?", "param_1": param_1, "some": some}
```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {"message": "What am I doing here?", "param_1": param_1, "some": some}

Sad Code
Sad Code

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {'message': 'Hello World!'}

@app.get("/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```
Sad Code

```python
from fastapi import FastAPI

app = FastAPI()

@app.get('/')
async def root():
    return {'message': 'Hello World!'}

@app.get('/why/{param_1}')
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```

4 pieces of advice
Listen carefully...
```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root(): return {'message': 'Hello World!'}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```
Goal-oriented approach - How-To Guides

"How-to guides are directions that take the reader through the steps required to solve a real-world problem. How-to guides are goal-oriented.

How-to guides can be thought of as recipes, directions that guide the reader through the steps to achieve a specific end."

Source: https://diataxis.fr/how-to-guides/
Sad Code

1st advice - How to guide

Simple Docs Setup

Code snippets on how to setup documentation for a Python code.

Requirements: Python 3.x

Installation

Please create your Python virtual environment and install the requirements, for example with venv:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements/app.txt
```
Sad Code

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {'message': 'Hello World!'}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```

Simple Docs Setup

Code snippets on how to set up documentation for a Python code.

Requirements: Python 3.x

Installation

Please create your Python virtual environment and install the requirements, for example with venv:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements/app.txt
```
Learning-oriented approach - Tutorials

"Tutorials are lessons that take the reader by the hand through a series of steps to complete a project of some kind."

Source: https://diataxis.fr/tutorials/
Sad Code

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {'message': 'Hello World!'}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```

2nd advice - Tutorial

**Tutorial - Basic Sphinx setup**

Follow the next steps:

- add Sphinx and recommonmark (Markdown support into your requirements file)
- install Sphinx package

```
pip install sphinx
```

- create a new docs directory in your project

```
mkdir docs
```

- go to docs directory

```
cd docs
```

- follow basic Sphinx setup by running

```
sphinx-quickstart
```

It's still too early to run build, we need to setup our configuration to make sure that we are able to generate code_reference.
I got another friend!

Sad Code

```python
from fastapi import FastAPI

app = FastAPI()

@app.get('/')
async def root():
    return {'message': 'Hello World!'}

@app.get('/why/{param_1}')
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```

Tutorial - Basic Sphinx setup

Follow the next steps:

- add Sphinx and recommonmark (Markdown support into your requirements file)
- install Sphinx package

```
pip install sphinx
```

- create a new docs directory in your project

```
mkdir docs
```

- go to docs directory

```
cd docs
```

- follow basic Sphinx setup by running

```
sphinx-quickstart
```

It's still too early to run build, we need to setup our configuration to make sure that we are able to generate code_reference.
Understanding-oriented approach - Explanation

"Explanation, or discussions, clarify and illuminate a particular topic.

... Explanation clarifies, deepens and broadens the reader’s understanding of a subject."

Source: https://documentation.divio.com/
3rd advice - Explanation

Why do we need documentation?

Explanation and motivation

Do you document your code? Do you think it is important?

Benefits of documenting your code:

- if you will get back to your code in 6 month, you will not become a detective to find out how this code works
- easy to onboard new team members
- easy to share knowledge
- if your code is open source - easy to start contributing
- easy to see purpose and motivation of each piece of code

I love this quote from The Documentation System:

There is a secret that needs to be understood in order to write good software documentation: there isn’t one thing called documentation, there are four.

They are: tutorials, how-to guides, technical reference and explanation. They represent four different purposes or functions, and require four different approaches to their creation. Understanding the implications of this will help improve most documentation - often immensely.
Sad Code

```python
from fastapi import FastAPI
app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {"message": "What am I doing here?", "param_1": param_1, "some": some}
```

And more friends!

Why do we need documentation?

Explanation and motivation

Do you document your code? Do you think it is important?

Benefits of documenting your code:

- if you will get back to your code in 6 month, you will not become a detective to find out how this code works
- easy to onboard new team members
- easy to share knowledge
- if your code is open source - easy to start contributing
- easy to see purpose and motivation of each piece of code

I love this quote from The Documentation System:

There is a secret that needs to be understood in order to write good software documentation: there isn’t one thing called documentation, there are four.

They are: tutorials, how-to guides, technical reference and explanation. They represent four different purposes or functions, and require four different approaches to their creation. Understanding the implications of this will help improve most documentation - often immensely.
Information-oriented approach - Reference

"Reference guides are technical descriptions of the machinery and how to operate it"

Source: https://diataxis.fr/reference/
Sad Code

from fastapi import FastAPI

code = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/why/(param_1)")
async def why_function(param_1: int, some: Optional[str] = None):
    return {"message": "What am I doing here?", "param_1": param_1, "some": some}

4th advice - Reference guide

Code Reference

- app package
  - Submodules
  - app.main module

Read the Docs
"Someone, who can teach you a lesson, but not a teacher.

Someone, who can guide you to a goal, but not a tour guide.

Someone, who can tell you everything about technical specs of your functions, but not an encyclopaedia.

Someone, who can explain you a particular topic, to help you to understand, but not Google."
Was it all about you?
Are you Documentation?

"In other words, what we call documentation is fundamentally not one thing, but four. Understanding the implications of this, and how those four different things work, can help improve most documentation."

Source: https://diataxis.fr/
Tutorial - Basic Sphinx setup

Follow the next steps:

Explaination and motivation

Do you document your code? Do you think it is important?

Benefits of documenting your code:
- if you will get back to your code in 6 month, you will not become a detective to find out how the code works.

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {'message': 'Hello World!'}

@app.get("/why/{param_1}")
async def why_function(param_1: int, some: Optional[str] = None):
    return {'message': 'What am I doing here?', 'param_1': param_1, 'some': some}
```

Happy Code!
Would you document your code?
• people forget things
• people leave the code alone
• new people come to contribute

Documentation is very important.
• choose main source/tool for documentation
• make sure that it's up-to-date
• https://diataxis.fr/

• https://understandlegacycode.com/blog/where-to-put-documentation/

• https://realpython.com/python-project-documentation-with-mkdocs/
How to start?

- start as simple as possible
- go to version controlled docs
Documentation for your code?

• add docstrings to your code
• use Sphinx, MkDocs to prepare documentation
• try Read The Docs, GitHub Pages
• add more documentation
Demo? 🤔

https://github.com/atymoshchuk/simple_docs_setup
Welcome to SimpleDocTutor’s documentation!

- Intro into Simple Docs Setup
  - Main idea
  - Tutorial - Basic Sphinx setup
- Why do we need documentation?
  - Explanation and motivation
  - Used sources and books to read
- Code Reference
  - app package

Indices and tables

- Index
- Module Index
- Search Page
You can not force people to read documentation.
I would love to hear back from you!

https://atymo.me/

https://github.com/atymoshchuk/simple_docs_setup

Thank you!