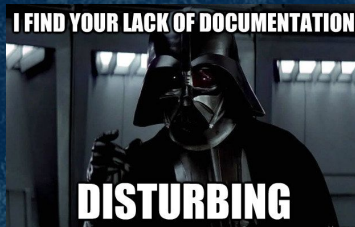


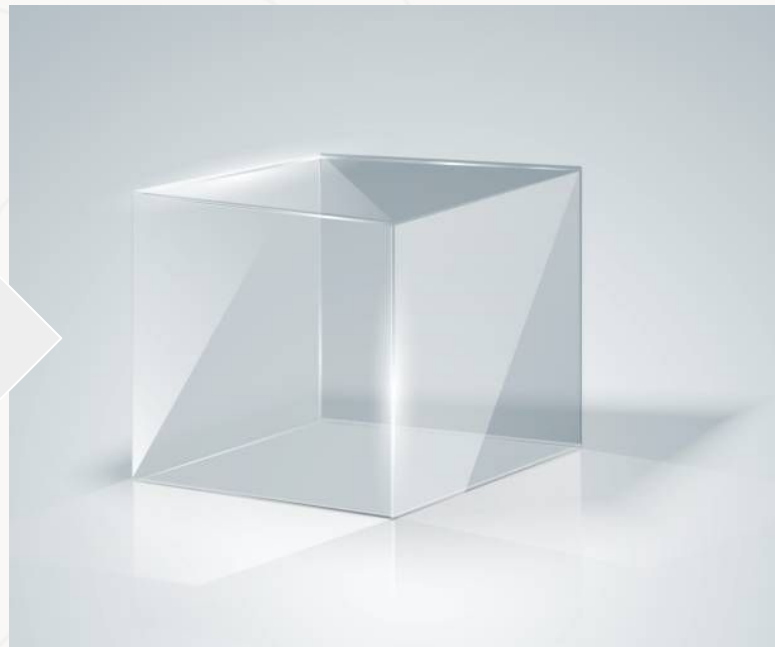
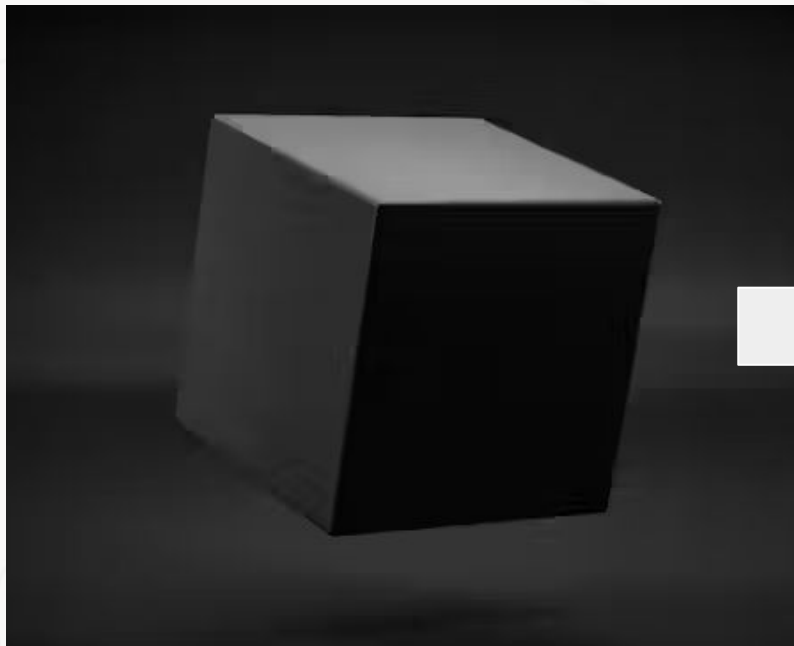
Write once, reuse often: How do you build sustainable code?

November 20, 2024
Danielle's & Magali's Docuvent



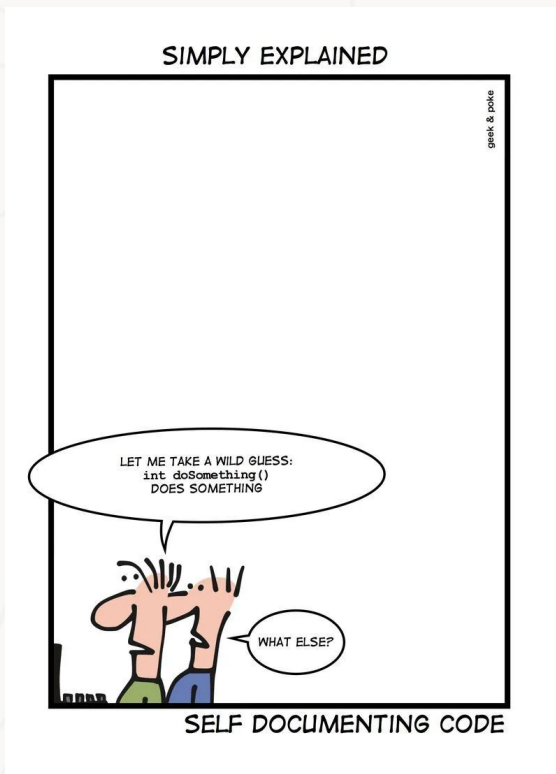


How to go from a black box to a glass box



Today's agenda

1. **Overview** of what documentation is about & why you might want to use it
2. **Best practices** of documentation: diving into specifics and items you can check off
3. A little collaborative demo **exercise & questions**



What is Documentation?

A recipe, manual, guidelines...



General Types of Documentation

Internal: for developers within an organization

External: for people outside an organization involved with the program

Low-level: within the source code and specifies specific lines/parts of code

- Comments, docstrings

High-level: the overall picture → code architecture, design principles

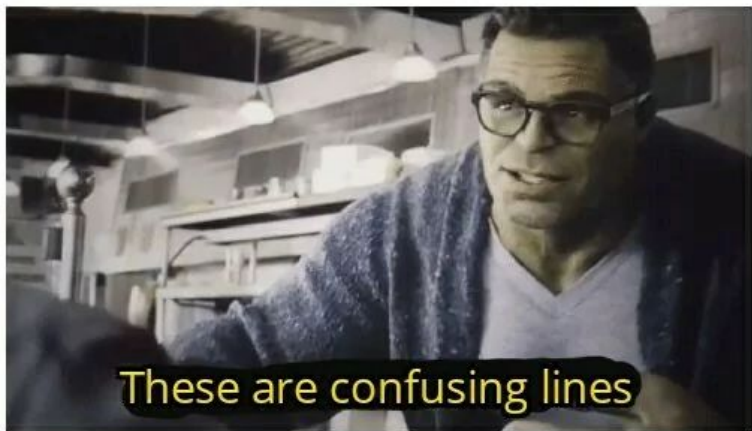
Walkthrough: “guided tour” → identify points of interest in code

- Link between low-level and high-level

Why documentation?

Me: writes code with
no documentation

Me: *one month later*



ProgrammerHumor.io

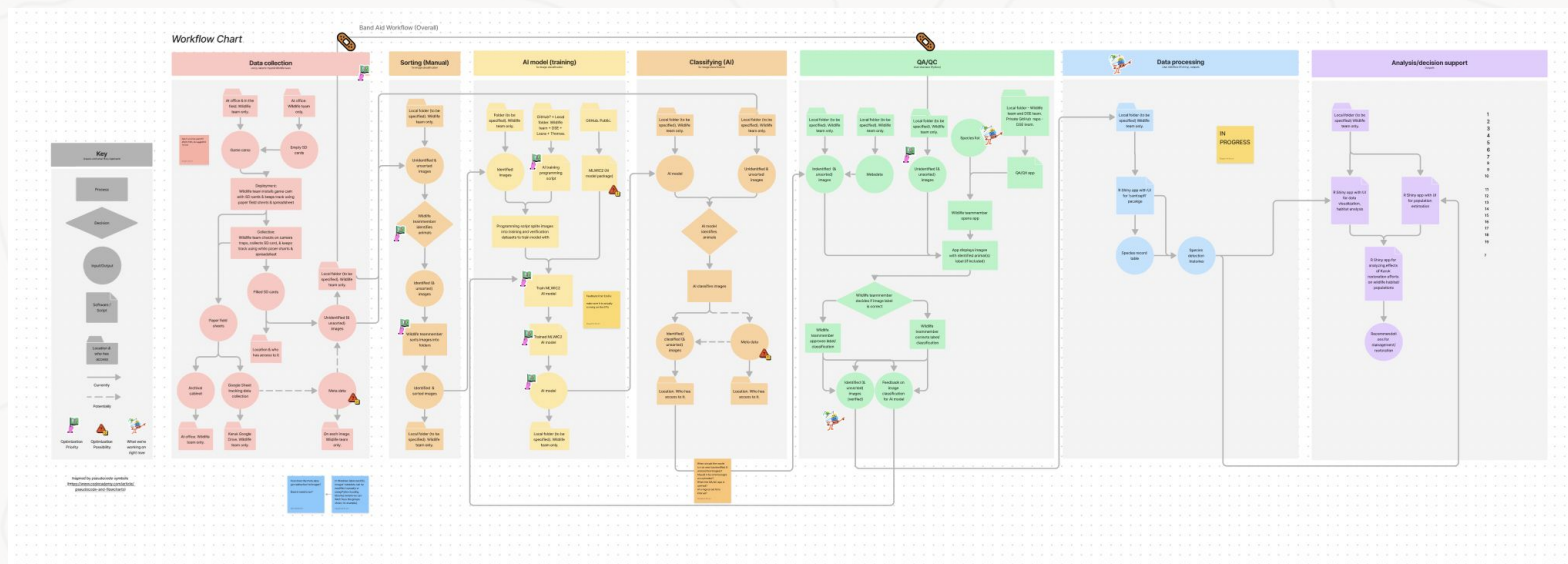
- Team projects / shared coding
- External audience
- For yourself: makes for better code, reference point, and much better debugging
- Knowledge loss and technical debt

How do you do documentation?

What are the best practices for documenting code?

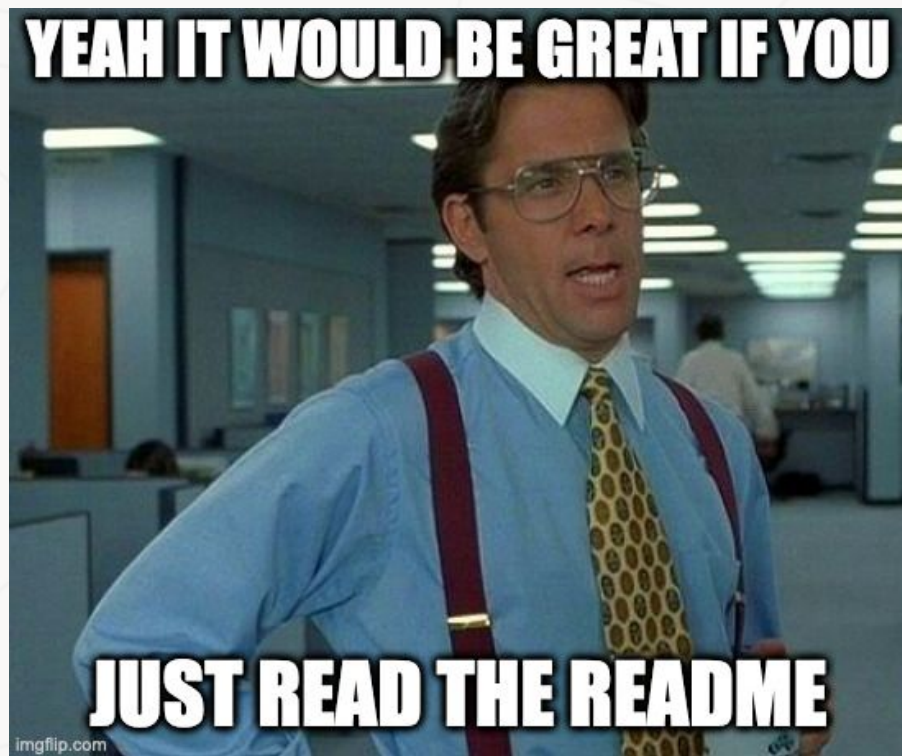
Design doc

- **Blueprints:**
 - Written/created before coding!
 - Planning, design process, initial feedback



README

- **R(ecipe)EADME.md**
 - What the project does
 - Why the project is useful
 - How users can get started with the project
 - Where users can get help with your project
 - Who maintains and contributes to the project
- Also typically includes:
 - License, contribution guidelines, and a code of conduct



Licensing

- What people can or can't do with your code
 - LICENSE.txt, LICENSE.md

The screenshot shows a GitHub repository named 'dse-nps-landing-page' (Public). The repository was generated from 'imfing/hextra-starter-template'. The file list on the left includes: .devcontainer, .github/workflows, .vscode, content, layouts/tool, .gitignore, .gitpod.yml, CNAME, LICENSE (highlighted with a red box), README.md, and go.mod. The 'LICENSE' file is selected, and its content is displayed on the right. The license is the MIT License, dated Copyright (c) 2023 Xin. The license text is as follows:

MIT License

Copyright (c) 2023 Xin

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Docstrings

- A string right after defining a function
 - Describes the purpose of the function
 - Defines **parameters** (input) and **returns** (output)
 - Uses three double-quotes: `""" Docstring """`

```
def get_date_retrieved(df):  
    """  
    Input: Dataframe with 'Image Name' column  
    Output: Dataframe with 'Date Retrieved' column  
    Explanation: This function extracts the date from the 'Image Name' column and stores it in a new column 'Date Retrieved'.  
    'Date Retrieved' corresponds to the date that the camera was retrieved from the field.  
    """  
  
    df['Date Retrieved'] = df['Image Name'].str.extract(r'\d{8}')  
    return df
```

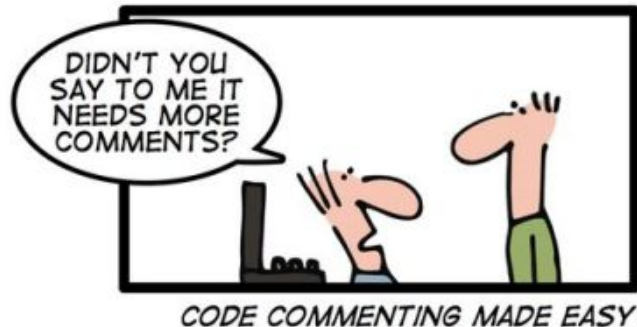
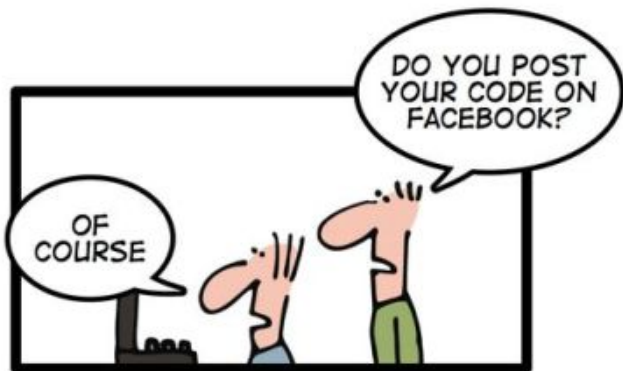
me trying to work out who didn't
write docstrings for the code



Comments

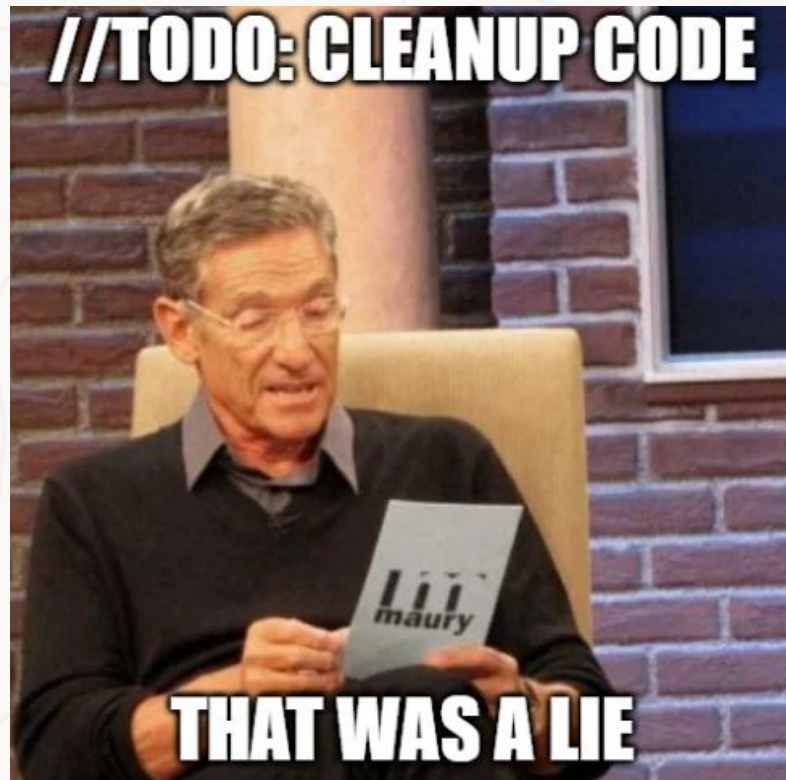
- Annotations throughout source code
 - #Comment, //Comment
 - Mainly used for debugging, modifications, questions, and reminders

```
# Data Frame: converting non-8-digit dates to date format  
def create_df_1(df):  
    df_1 = df.copy()
```



TODO and FIXME notes

- TODO: parts of code to work on eventually
 - Adding features, manage progress
- FIXME: reminders of broken code to fix
 - Typically code that is not urgent to fix
- Should **NOT** be present in stable code



Type hinting

- Indicates expected data types for function arguments and return values
 - String, float, integer, boolean, double, dataframe, etc.

This is how you annotate a function definition

```
def stringify(num: int) -> str:  
    return str(num)
```

And here's how you specify multiple arguments

```
def plus(num1: int, num2: int) -> int:  
    return num1 + num2
```

If a function does not return a value, use None as the return type

Default value for an argument goes after the type annotation

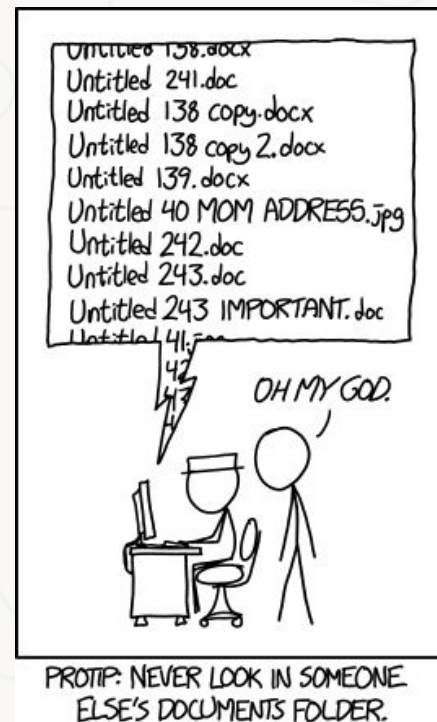
```
def show(value: str, excitement: int = 10) -> None:  
    print(value + "!" * excitement)
```

Naming conventions

- For data: files
- For code: variables, functions, classes
- **Best practice:** establish beforehand!

👉 Let's take a look at this [excellent worksheet](#)

- Put it in the README.md!
- **Avoid:** ecotech_last_draft_final_FINAL.ppt
- **Do:** use self-explanatory title, include (numerical) versioning (e.g. by date)
- Example: ecotech_20241120_v3.ppt



Naming conventions: functions, variables, classes

doSomething(folder_path)

vs.

count_jpg_files(folder_path)

var1

vs.

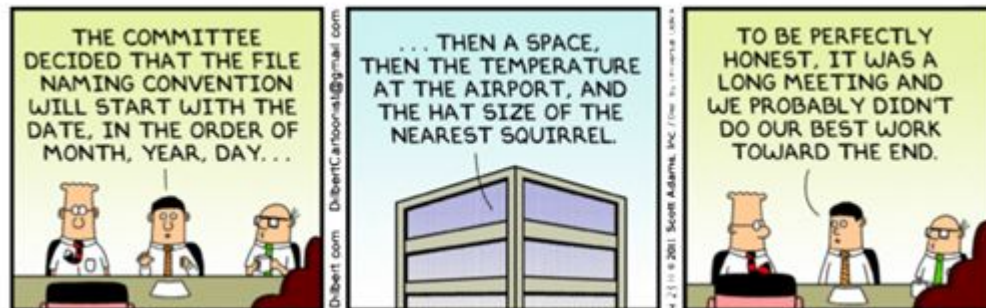
count

Model



vs.

Counter

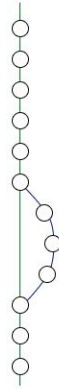
```
def count_jpg_files(folder_path):  
    jpg_count = 0  
    for file in os.listdir(folder_path):  
        if file.lower().endswith('.jpg'):  
            jpg_count += 1  
    return jpg_count
```



GitHub versioning: commit messages, PR requests


 **Victor Timi**  [@victor__timi](#)





A clean code with terrible commit messages is like preparing a wedding cake with no icing [#git](#) [#coding](#)

	Comment	Date
	WIP	3 days ago
	Off for lunch	1 day ago
	End of code for today	20 hours ago
	I am tired AF	18 hours ago
	Happy Weekend Team	16 hours ago
	First to commit	14 hours ago
	Fixed final bug	10 hours ago
	Added a new feature	9 hours ago
	Fixed another bug	7 hours ago
	Made some changes	5 hours ago
	fixed two build-breaking issues	3 hours ago
	Bugs are never ending, fixed another bug 🙄	2 hours ago

1:27PM · Oct 4 2022 · [Twitter for iPhone](#)

1.1M 1,240 5,579 3,987

 **Magali de Bruyn** Add datetime to the print statements (for the log file) 3e6e269 · 4 months ago

 .gitignore	Add shell script for successful cron job execution with virt...
 README.md	Add shell script for successful cron job execution with virt...
 example.env	Initialize repo & push code
 requirements.txt	Update virtualenv / venv instructions + dependencies

Testing out a pull request for visualizations #3

 Merged [danilouie](#) merged 25 commits into [main](#) from [prototype-viz](#) on Aug 13

 Conversation 5  Commits 25  Checks 0  Files changed 8



danilouie commented on Aug 8

Most of my work is in the notebook `interactive_viz.ipynb` where there is an interactive countplot, scatterplot, and barplot. Currently, the plots are able to take in user input about what species they want to look at, and the scatterplot can group the count by day, week, or month.

Things to develop:

- setting colors/shades for species (currently, the colors on the graph sometimes adjusts for each species, although it is properly reflected in an updated legend)
- adding an option for users to look at a specific timeframe (like August 2020-August 2021)

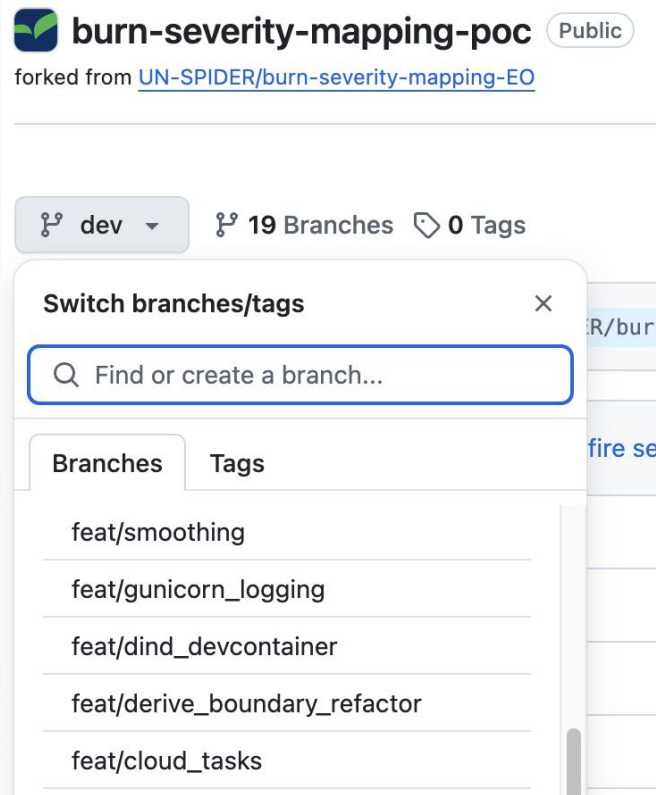
 **danilouie** and others added 25 commits 5 months ago

 update environment file

76e86fb

GitHub versioning: branches naming convention

- Preceding category
 - feat/ or feature/
 - bugfix/
 - test/
- Short description
 - feat/smoothing
 - feat/responsive-ux



Configuration files

- YAML, XML, JSON, requirements.txt ... environment files
- List external frameworks, libraries, and dependencies



name: geoml38

channels:

- conda-forge
- apple

dependencies:

- python=3.8
- pip
- ipykernel
- numpy
- pandas
- xarray
- geopandas
- jupyterlab
- geojson
- affine
- rasterio
- scikit-learn
- earthengine-api
- matplotlib
- seaborn
- tensorflow-deps
- unicode
- geemap
- pip:
 - radiant_mlhub
 - tensorflow-macos
 - tensorflow-metal

This lends itself to object-oriented programming...

Object-oriented programming as a documenting structure

```
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

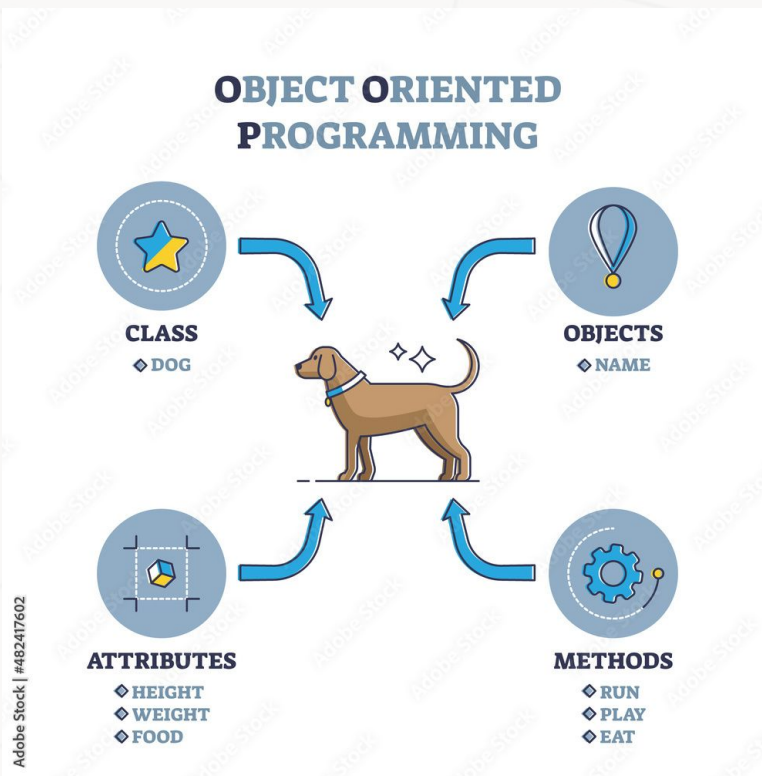
    def bark(self):
        print("Woof!")

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age

    def get_breed(self):
        return self.breed

dog1 = Dog("Fido", 3, "Labrador")
dog2 = Dog("Buddy", 5, "Golden Retriever")
```



Exercise time!

Can you document this code better?

Clone this repository in our [Github](#)

```
# minutes
def beep_boop_vrrrr(x):
    if x < 1:
        return "short"
    elif (x >= 11 & x < 3):
        return "medium"
    elif (x >= 3 & x < 5):
        return "long"
```

Answer key: Who's the potato king/queen? 🥔👑

- +1 if you renamed functions
- + 1 if you renamed variables
- + 3 if you added docstrings
 - + 1 if you added an explanation for what the function does
 - + 1 if you added what kind of argument is inputted
 - + 1 if you added what is the result of the function
- + 1 if you deleted unnecessary comments
- + 1 if you added type hinting
- + 10 if you figured out what this class is trying to do
- What else? What might we have missed? +x 😊