Documenting software
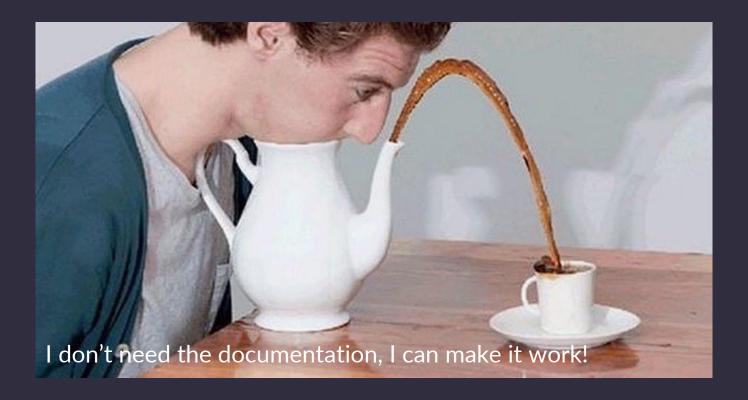
WRITE DOCUMENTATION THAT CONVEYS INFORMATION EFFICIENTLY AND INTUITIVELY

I don't need the documentation, I can make it work!

Why even bother with documentation…?

❑ Because you want people to use your code! But…
  ▪ If people can't figure out what is the purpose of your code, how to install your code, or how to use your code, they won't use it! Sounds obvious, right? **Because it is.**

❑ Leaning a software by its source code is tremendously inefficient
  ▪ *"There is a **vanishingly small** number of people who will source-dive to use any code out there, compared to the people who will use your code when properly documented."* ~ Eric Holscher, co-founder of readthedocs and writethedocs
  ▪ Actually, for complex software it is literally impossible to learn it from source code alone!

❑ Good documentation, like good code, will motivate contributions!
  ▪ You can only get contributions after having users, which happens *after having a documentation*

❑ **Documentation is the most important part of a software if you want >1 user count.**

You get a tech job

You ask for the documentation for an internal project

They tell you "there is no documentation, just look at the code"

The code has no comments

The variable names are 3 letter long acronyms

Most files are 2000+ lines of code

sounds familiar...?

Documentation

❑ What's the role of documentation?

1. Teach new users the software

2. Serve as reference for existing users

3. Define the public API (how to use the software) → also defines what is a bug or not! (remember: only the *intended* behavior of software can be described!)

❑ A *good documentation* will make the user:
- Understand what this software is about
- Get an immediate overview of the features of the software
- Find the information they need fast
- Find it where they expect it to be
- Understand the information quickly
- Have no ambiguity after reading the docs (no guessing!!!)

The Markdown syntax

❑ We will be writing documentation using Markdown, so we better learn the basics!

```
# Markdown intro
Markdown is a way to style text by using plain text that is later rendered into HTML.

## Basics
Text prefaced with one to six `#` and then an empty space creates headers.
- Markdown can do lists like the one you read now with `-`
- Make text **bold** or _italic_, and can even do [links](www.google.com)
- Markdown can also do math with $ and $$, once rendered, depending on renderer.

1. Markdown can also do numbered lists, like the one you read now.
2. Tables, in-line `code` (backticks), code blocks (three backticks), and many more!
3. See https://www.markdownguide.org/ for more syntax guidance!

Here is a code block with Julia highlighting:
```julia
rand()
x = 1 + 1
```
```

pro tip: this is the syntax GitHub uses as well, you can use while commenting on GitHub!

Documentation strings

❑ ***Documentation strings*** are descriptions of what a function does and how it can be used, and form the basis of a scientific software documentation
  ▪ They are written in human language using Markdown, but they are *not* comments
  ▪ They are *superior* to comments and should be preferred for explaining things when possible

❑ They are accessible from within the language, interactively. Example.
  ▪ In Julia they are the topmost string above the function, in Python they are the first string in the function

❑ Public API should be defined entirely on the docstrings of the exported names

❑ ***First write the docstring, then write the function (imagine what it should do)***

Writing good documentation strings

❑ Example: Lyapunov exponent function from DynamicalSystems.jl
  ▪ Docstring in the source code
  ▪ Docstring expanded/rendered in the documentation

❑ Documentation string structure according to DynamicalSystems.jl:
  1. Clear call signature in code syntax, including expected input types if necessary
  2. Brief summary of the function
  3. [Optional] Return value and type if not obvious
  4. [Optional] References to related functions if sensible
  5. [Optional] Keyword arguments list if the function has some
  6. [Optional] Detailed discussion of functionality if function behavior is scientifically involved
  7. [Optional] Citations to relevant scientific papers

❑ Clear distinction between code and human-language-text using backticks `. E.g., when referring to something in the call signature, use code formatting explicitly!

❑ Examples extremely useful! Best showcased via runnable code executed during documentation generation!

Exercises: Documentation strings

❑ Write documentation strings for the generalized code for temporal means
  ▪ File `running_means_generalized.jl` from block 3

❑ Fully leverage markdown syntax, and reference related functions!
  ▪ In Julia this is done as `[`function_to_ref`](@ref)`

❑ *Max time: 15 minutes!*

Skeleton of the documentation pages

❏ After you have the docstrings, you can use them to make a full documentation that can be hosted online… But what should be in your documentation?

❏ It is as easy as 1-2-3

1. Homepage: first thing any user will see

2. API declaration: literally expanding the documentation strings with guiding/connecting sentences when necessary

3. Tutorial(s) / Examples: educative usage of the code base

What to include in the homepage?

1. Logo!
   - Makes your software unique, memorable.
   - Shows you care for it enough to give it a "face".

2. Introductory paragraph
   - must capture, in only a couple of sentences, what is this software about.

3. Getting started
   - how to install, preliminary knowledge, how to navigate the documentation

4. List of features (if too extensive, can be a dedicated page)

5. Design philosophy (optional)
   - More in-depth description of the software design, what makes it special
   - Can also include a comparison with competing software

6. How to contact the authors / ask questions / get involved in development

Depth of exposition

❑ Good documentation: *various levels of depth of exposition*

❑ Highest level: Summary of what the software does

❑ High level: One overarching tutorial on the core of the software

❑ Intermediate level: Examples that showcase end-to-end usage of specific or advanced functionality

❑ *Examples must be RUN. Not written!*
  ▪ Tremendously lowers the risk of broken / un-updated docpages!
  ▪ Specific language packages provide tooling for achieving this, see later slides

❑ Lowest level: Declaration of pubic API / docstring expansion.

Documentation example: Agents.jl

❑ Docs of Agents.jl have been celebrated many times,
so let's see how they satisfy the above tips

❑ https://juliadynamics.github.io/Agents.jl/stable/

❑ Specially highlight:
▪ Structure of index page (home page)

▪ Dedicated tutorial

▪ API page and its relation to the side bar

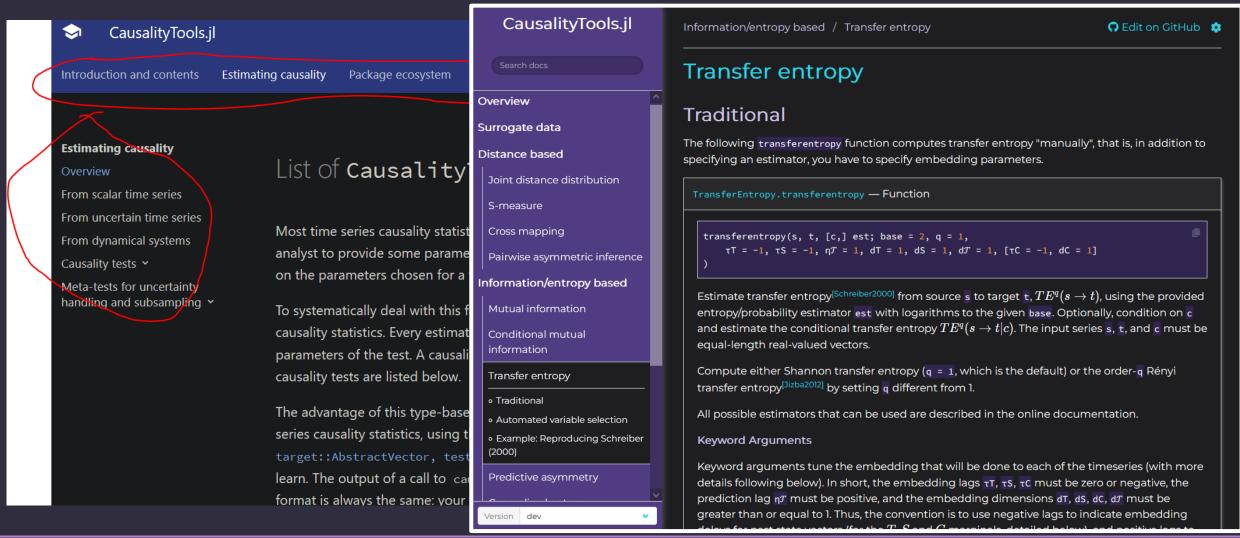▪ Examples are run, not written (go to an example, click "Edit on GitHub")

ToC: Table Of Contents

❑ ***The Eye Horizon:***
- ▪ The viewer will scan the page content currently in front;
- ▪ This scan should indicated if the information the user needs is in this page

❑ The Table of Contents is the best way to satisfy the "eye horizon" problem!
- ▪ Requires you to structure the doc page around headers and subheaders

❑ Get header ordering right!
- ▪ Subheaders must really be subtopics of the upper level
- ▪ Same level of information variety must be same header level
- ▪ Sweet-spot balance between overview and information exposure

❑ All ToC information must be in the same place (a.k.a. proximity)
- ▪ Do not split ToC information to the left, right, or top of the layout. All in the same place!

ToC: Proximity illustration

❏ From a discussion on how to improve the docs of CausalityTools.jl

The one true ~~Morty~~ Documentation

❑ You want to eliminate chances of "forgetting to update something". Two rules:

1. Documentation **must exist only at one place**
   - GitHub webpage hosting, READMEs, GitHub Wiki, Private Webpages, GitLab pages, …
   - **NO!** Keep it in **ONE** place, coherently, and allowing cross-referencing everything!
   - I am against having anything in the README.md besides the "opening paragraph"!
   - Clarification: for small projects, the README is the entire documentation

2. Thing X must be **defined only once, at one place**, and explained at that place in your documentation with as much detail as necessary
   - All other parts of the documentation that use X must cross-reference its definition
   - Not explain it again! Avoid duplicating information!
   - Actually cross reference via the documentation pipeline, so actual hyperlink!

PRAISE THE ONE TRUE ~~MORTY~~ DOCUMENTATION

Use unique identifiers

❑ Also known as "Pick One Word Per Concept"
  ▪ *Clean Code – A Handbook of Agile Software Craftmanship*

❑ When referring to the same things, use the same words. Exactly the same.
  ▪ This is a good advice for any form of technical writing (paper, docs, code)
  ▪ The same advice therefore also applies to naming variables in your code!

❑ This establishes **an unambiguous connection** between the English words, and the object they refer to, which is typically code

❑ It is much more important for a technical document to be clear and precise, than to be poetic. **Do not optimize word variety, optimize word clarity.**

❑ E.g.: Fetch, Retrieve, Get, Obtain… Pick one!

❑ E.g.: Sketch, Diagram, Concept, Idea… Pick one!

Actually writing documentation pages

❑ Each language has some packages that help create documentation...
- Expand documentation strings
- Allow cross referencing documentation strings
- Creating HTML pages out of markdown documents
- Additional markdown decorators like enabling math or running code in the docstring

❑ Then GitHub can be used to host the documentation via CI

❑ Julia recommended tool: Documenter.jl - https://juliadocs.github.io/Documenter.jl

❑ Python recommended tool: Sphinx - https://www.sphinx-doc.org

❑ It's beyond the timetable of the workshop to teach these tools in detail...

❑ But a working template for each exists in the default repositories we've shared

Creating & hosting a documentation

❑ Let's go through the default template and demonstrate how to make some documentation pages, auto-generate them and host them online

❑ Reminder, default project templates are in:

❑ https://github.com/JuliaDynamics/ScienceProjectTemplate Julia

❑ https://github.com/JuliaDynamics/example-python Python

Your code → documented

❑ Create a documentation for your code that is also hosted online ☺

❑ Remember our tips on good docs
 ▪ (scroll up the slides, I won't repeat them here!)

❑ How to approach this exercise:

❑ The most important thing by far is to add docstrings to your functions in your `src` that are of general use, and you expect others to have use for

❑ Then, write an "outline of your documentation", a simple `index.md` markdown file, that follows our tips on "a good homepage"

❑ Then, try to expand documentation strings at the end of this file, following the syntax of the documentation generator (see example repos)

❑ Then, try to generate the documentation locally by running the documentation generator (again, the example repos have this set up)

❑ And only as a last step, worry about hosting the documentation online.

❑ *Remember: most important part of the docs is the documentation strings!*