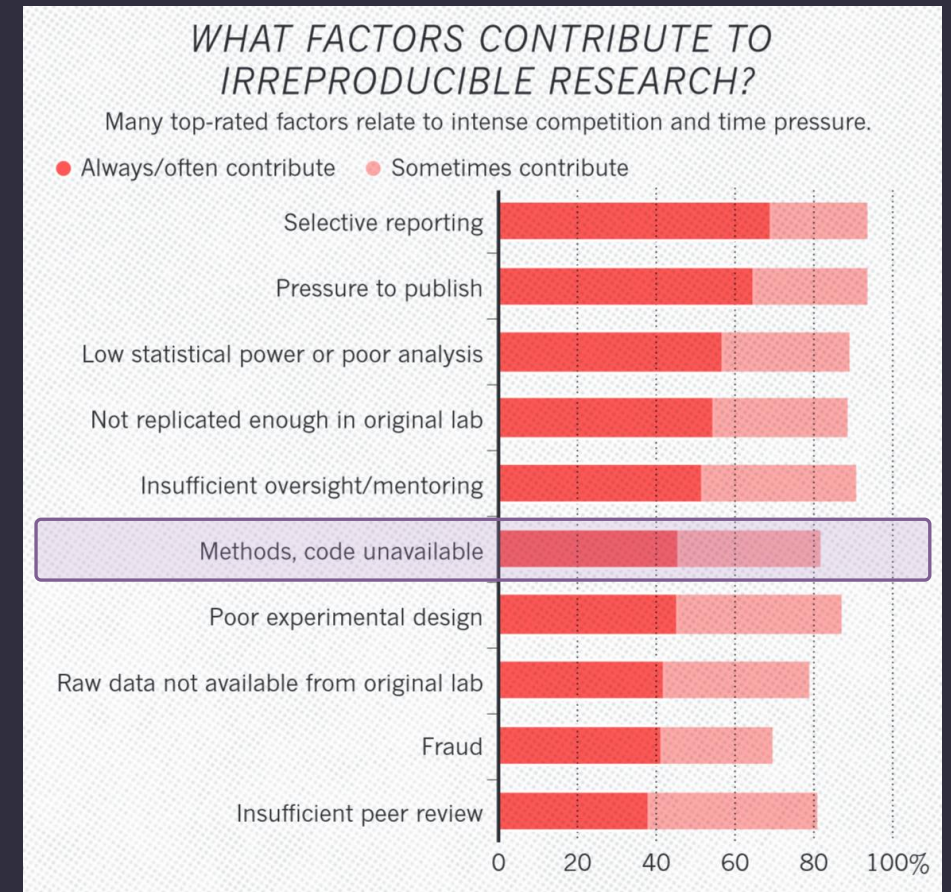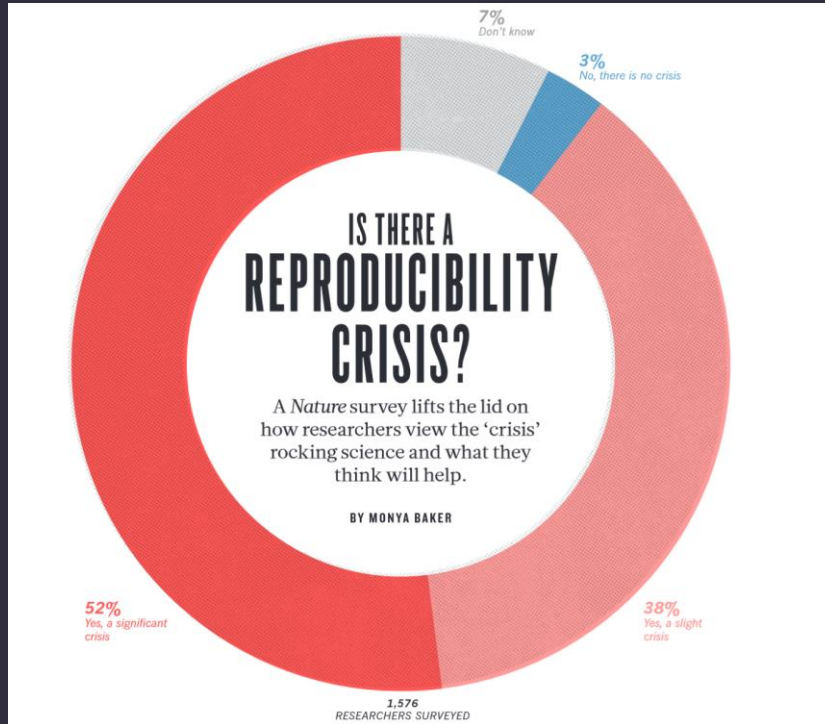# Scientific project reproducibility

PUBLISH REPRODUCIBLE PAPERS,
END THE CYCLE OF DESPAIR

Is there a reproducibility crisis…?

❏ 1,500 scientists lift the lid on reproducibility
  ▪ Nature 533, 452–454 (2016)

❏ *"Reproducibility is like brushing your teeth, once you learn it, it becomes a habit"*
  ▪ Actually, this can be said for literally everything you learn in this workshop…

Science & code: a story of love & hate

❏ Modern computer-based science *needs* code. So, it's part of science, right…?

❏ Oddly, for whatever god-forsaken reason, code is almost never mentioned…

equations/algorithm descriptions

Majestic algorithm
$$f = 5x^2 - 3$$
$$g = 12$$
$$r = g + f/2$$

Takes effort to derive.
Takes no effort to share.
Human error can be in any line.
Researchers use other people's equations.
Only a description of derivation of r

→ **Always in the paper, clearly is "science"**
   We do not trust, we see and check

code

```
function majestic(x)
    f = 5x^2 - 3
    g = 12
    r = g + f/2
end
```

Takes effort to write & more to optimize.
Takes little effort to share.
Human error can be in any line.
Researchers use other people's code.
Actually produces the value of r

→ **Not in the paper, not "science" (???)**
   We don't see, we must trust. Why?

Reproducibility = shared code

❑ The only way for 100%, absolutely true reproducibility is to share code

❑ A human-language based description of code is *not enough*

1.8 Footprint strength versus evolutionary conservation

We additionally calculated the maximum phyloP evolutionary conservation score over the same set of footprints. The maximum score was derived over the core footprint region (no buffering), with 10% of outlying scores removed. As before, footprints were ordered by their FOS values, and signal data were plotted using loess curve fitting with a span of 25%. We applied a linear regression model with R statistical software (http://www.r-project.org) collecting the associated F-test's P value.

from Neph, Shane, et al. "An expansive human regulatory lexicon encoded in transcription factor footprints." Nature 489.7414 (2012): 83. (Supplement)

Saying "we used Python to do our analysis" is as useful as saying "we run the analysis on a computer"

❑ Furthermore, some variable's value may be forgotten in text.
But code, *by definition, will not run unless all variables have a value!*

❑ Please, always share your code! Even if it isn't good code, *it still makes a big difference*

Reproducibility ≠ replicability

❑ To clarify: here we focus on a very small aspect of replicating a scientific study:

❑ Given exactly the same input, produce the same output. *Compute the same things.*

❑ Obviously, replicability is important, but also much harder
  ▪ You need to describe your methods well
  ▪ And their caveats and limitations
  ▪ But because this is a coding workshop, this goes off-topic

❑ Don't think "I've described my methods so I don't need to share code".

❑ Please share code.

❑ *Please share the freaking code.*

Principle of least gears

❑ Bad:
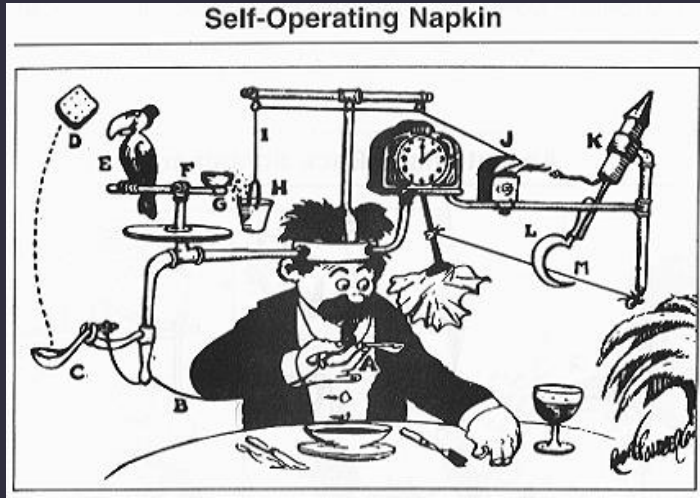
Good:

❑ Rule of thumb: *minimize the different gears used in your project*

❑ If you can use a *single* programming language/environment for *literally everything* in your project, from submitting jobs to cluster to generating figures, *then do it*
- E.g.: use bash to submit to cluster, whatever library to plot Earth, Python to analyze
- Just use Python! Reader has to (a) only read a single language and (b) only run a single language
- Yes, you can submit jobs to a cluster with Python, you don't need to learn bash

DrWatson

❑ DrWatson is a *scientific project assistant* software.
- ▪ Full disclaimer, I'm the lead developer, so this is partly an advertisement

❑ It is a package for Julia, that is used within the language (no external tooling)

❑ Among other things, it helps you with:
- ▪ making self-contained reproducible projects
- ▪ activating projects on run-time and enabling local pathing from root
- ▪ automatic encapsulation of git status in saved output files
- ▪ automating project input-to-output pipelines using configuration files
- ▪ preparing ensemble simulations
- ▪ collecting and filtering simulation output files

❑ Last time we checked, no real alternatives, but similar packages in other languages:
- ▪ Cookiecutter (Python), Sumatra (Python/R/Matlab), ReproZip, explore (Matlab), recipy (Python)

Project structure

❑ make the habit of following a specific structure for all your science projects

❑ Here I am showing the default of DrWatson, which I use in *exactly the same form* in *all* my projects

❑ Many project-template-providing packages exist in all languages. The exact structure is not as important as having one and sticking with it

```
projectdir              <- Project's main folder. It is initialized as a Git
                           repository with a reasonable .gitignore file.

├── _research           <- WIP scripts, code, notes, comments,
│   │                      to-dos and anything in an alpha state.
│   └── tmp             <- Temporary data folder.

├── data                <- **Immutable and add-only!**
│   ├── sims            <- Data resulting directly from simulations.
│   ├── exp_pro         <- Data from processing experiments.
│   └── exp_raw         <- Raw experimental data.

├── plots               <- Self-explanatory.
├── notebooks           <- Jupyter, Weave or any other mixed media notebooks.

├── papers              <- Scientific papers resulting from the project.

├── scripts             <- Various scripts, e.g. simulations, plotting, analysis,
│   │                      The scripts use the `src` folder for their base code.
│   └── intro.jl        <- Simple file that uses DrWatson and uses its greeting.

├── src                 <- Source code for use in this project. Contains functions,
│                          structures and modules that are used throughout
│                          the project and in multiple scripts.

├── test                <- Folder containing tests for `src`.
│   └── runtests.jl     <- Main test file, also run via continuous integration.

├── README.md           <- Optional top-level README for anyone using this project.
├── .gitignore          <- by default ignores _research, data, plots, videos,
│                          notebooks and latex-compilation related files.

├── Manifest.toml       <- Contains full list of exact package versions used currently.
└── Project.toml        <- Main project file, allows activation and installation.
                           Includes DrWatson by default.
```
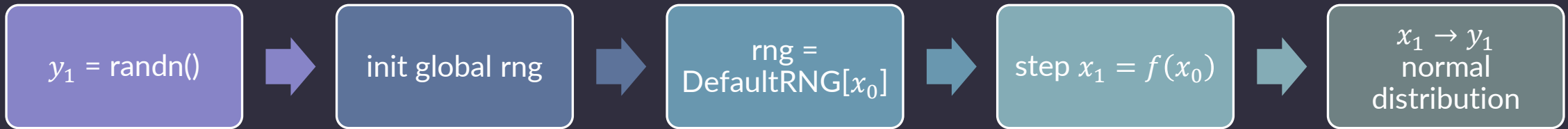
Use notebooks appropriately

❑ Notebooks (Jupyter, Weave, Pluto) are a fancy new way to code that makes a mixed medium interlacing code, markdown, and produced figures

❑ ***Notebooks are good for presentation, but not for creating a codebase***

❑ Do not abuse Notebooks for doing research!
- Notebooks (with the exception of Pluto) are full of hidden variables
- Notebooks do not communicate well with each other
- Notebooks are hard to update: you need to run them through their server
- It is really difficult to version control notebooks
- Scripts are cleaner and train you to use a source folder structure

❑ Suggestion: use notebooks for structuring reports/drafts, plotting scripts, or presentations (if you use a notebook for a week or so, drop it!)

❑ Use a high quality IDE like VSCode for day to day work and establish a code base
- Modern IDEs are *truly powerful*
- *In fact, you can create Jupyter notebooks entirely within VSCode*

Datastructures: keep em' simple

❑ ***Avoid creating new data structures, use native ones as much as possible.***
- ▪ You want a named parameter container? Use a dictionary
- ▪ Use the simplest possible data structure that does what you need
- ▪ Favor existing data structures from existing packages than rolling your own
- ▪ ***I.e., don't reinvent the wheel please!***

❑ Similarly: don't create new classes for 1 task that could be a function instead!

❑ Use special/new data structures for:
- ▪ performance optimization
- ▪ multiple dispatch
- ▪ algorithm deciding arguments (e.g. sort(x; alg = QuickSort())
- ▪ name space establishment when developing a software
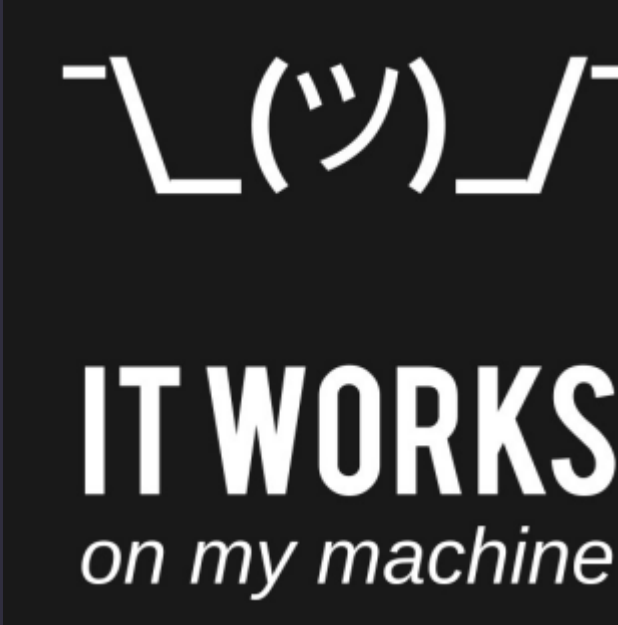- ▪ developing interfaces for a new API

Deterministic code

❑ Computers are deterministic → you should write ***deterministic code***
  ▪ For this, you need to understand random number generators (RNGs)

❑ RNGs, with the exception of true entropy sources, are not actually random
  ▪ Some of them are chaotic maps, $x_{n+1} = f(x_n)$ with f the RNG function
  ▪ RNGs generate sequences $X = \{x_0, x_1, \ldots, x_n\}$ which are (typically) distributed uniformly and sufficiently randomly in the unit interval [0, 1).
  ▪ X can be transformed to any distribution Y via the inverse of the cumulative density function

| y_1 = randn() | ➡ | init global rng | ➡ | rng = DefaultRNG$[x_0]$ | ➡ | step $x_1 = f(x_0)$ | ➡ | $x_1 \rightarrow y_1$ normal distribution |

❑ If you use RNGs, you should explicitly set their seed always!
  ▪ Default seed typically depends on system clock, temperature, etc.
  ▪ It is even more transparent, and better performance, to explicitly use an `rng` object.
  ▪ E.g., Julia, instead of `x = rand()`, do `using Random; rng = Random.Xoshiro(seed); x = rand(rng)`.

Before sharing: package versions

❑ Often: share a project and then "nothing works"

❑ You want to detach your project from your machine as much as possible... but how?

❑ Your project must be bundled with a list of versions of every package you have used (and their dependencies)
  ▪ How to do this? Only language-specific solutions, see next slide

❑ (yes, your own code should be versioned as well)

you want to absolutely avoid this hell-hole

❑ Another reason things work only on your machine: absolute paths
  ▪ Avoid using absolute paths in your scripts! Try to get to root folder and use relative from there.
  ▪ Or, use e.g. `DrWatson.srcdir` instead

Next level of "but it works on my computer!"

file:///C:/Users/[REDACTED]/Downloads/8021_w21_ms_21.pdf   4:46 PM

Good evening students, please go through the Answer keys for yesterday's
Revision paper sent on [REDACTED] on Friday. 👆   4:47 PM

The SemVer2.0 versioning system

❑ A version is a ***unique, unchangeable*** identifier of a code base's contents
  ▪ Establishes reproducibility, compatibility, and (tries to) eliminate dependency hell
  ▪ Any software used in published research must have an associated published version. If not, it is like citing "the conversation you had at the bar with that dude".

❑ ***Semantic Versioning 2.0.0*** is a well-established standard for versioning software. Given a version number vMAJOR.MINOR.PATCH, increment the:
  1. MAJOR version when you make backwards-incompatible API changes,
  2. MINOR version when you add new functionality in a backwards compatible manner, and
  3. PATCH version when you make backwards compatible bug fixes.
  ▪ Example: code using v1.2.1 is compatible with versions [1.2.1, 2)
  ▪ Example: code using v1.4.2 is NOT compatible with v1.3 because of (potentially) new API

❑ All software should have a CHANGELOG.md with noteworthy changes!

One project = one environment

❑ Most programming languages allow the concept of "environments"
- They are self-contained "spaces" where one can add packages and track package versions and specify compatibility bounds to packages
- Once "activated", the packages loaded on `import` statements are the versions specified
- They can be shared with other users and instantiated in other computers
- They allow for reproducible code by ensuring everyone runs the same versions
- By default, there is the "global environment" which is what is activated when the language starts.

❑ ***Each project should have its own environment!***
- This allows confidence that each project doesn't break over time due to updates elsewhere

❑ To create or activate an environment and add packages to it:

Julia
```
using Pkg; Pkg.activate("path")
Pkg.add("PackageName")
```

Python
```
conda env create -n <NAME>   # create a new environment
conda activate <NAME>   # switch to an existing environment
conda install <PKGNAME>    # add a package
```

Bundling project with package versions

❑ Julia: a file `Project.toml`:

```
name = "ScienceProjectTemplate"
[deps] # Specifies dependencies (no compat bounds)
Documenter = "e30172f5-a6a5-5a46-863b-614d45cd2de4"
DrWatson = "634d3b9d-ee7a-5ddf-bec9-22491ea816e1"
[compat] # Specifies versions
DrWatson = "2.9.1"      # Default: SemVer compatible
Documenter = "≥ 0.27" # 0.27 or upper, any
julia = "1.7.0"
```

❑ Julia's package manager generates a `Manifest.toml` file based on the above with exact versions of all dependencies

❑ To reproduce an environment: get the Project.toml containing folder, and then run:

```
using Pkg; Pkg.activate("path/to/Project.toml")
Pkg.instantiate()
```

❑ Python: a file `environment.yml`:

```
name: my-awesome-environment
dependencies:
    - python >= 3.9
    - matplotlib >= 3.0.0
    - numpy >= 1.21.6
    - pytest
```

❑ A machine file with all exact versions can be made with `conda list -e`

❑ To reproduce an environment from the `environment.yml` run:

```
conda env create -f environment.yml
```

How I make a 100% reproducible project

1. Initialize a DrWatson project and add all packages I foresee using

2. Open three columns in my IDE: source, script, console

3. Start doing the analysis on middle column, the script, which uses DrWatson

4. Functionality of general purpose goes to the source file after creation in script

5. Periodically the code base is reorganized to more script and source files

6. When ready to publish:
   - Make one script per paper figure or table that needs numbers
   - Publish the entire code repository (+ README, Project.toml, Manifest.toml) on GitHub
   - Assign a DOI to the GitHub repo via Zenodo.org. Then, cite the DOI in the paper!
   - Example: Minimal recipes for global cloudiness (latest paper with code base)

Your code → reproducible project

❏ Ensure you have an intuitive folder structure (source, scripts, plots, data, ...)

❏ Ensure you only use local/relative directories in the codebase

❏ Bundle your project with the appropriate environment
  ▪ How does your language stores which package versions you used?
  ▪ How do you bundle this information with your project?
  ▪ How does someone that wants to reproduce your figure "instantiate" this environment?
  ▪ Have this info in your README!

❏ Your exercise partner pulls latest version from GitHub and tries to reproduce it
  ▪ Do not upload large datasets on GitHub. Instruct in the README where to find it.

Your code → released and citable with DOI!

- ❏ You are ready to release the first version of your code!

- ❏ Hell, you can even attach a Digital Object Identifier to it!

- ❏ Here's how! (I'll show steps online as well)

- ❏ Create a Zenodo account at Zenodo.org and link your GitHub account there!

- ❏ Navigate to the repository of interest and "turn on" the DOI badge button!

- ❏ Go back to GitHub and create a new "Release"!
  - ▪ Typically first running release has version 0.1.0

- ❏ Go back to Zenodo, and you'll see you now have an associated DOI with this release!

- ❏ Transform it into BibTeX with https://www.bibtex.com/c/doi-to-bibtex-converter/ !

- ❏ Cite it in your paper! MUCH SUCCESS MUCH WOW!!!!
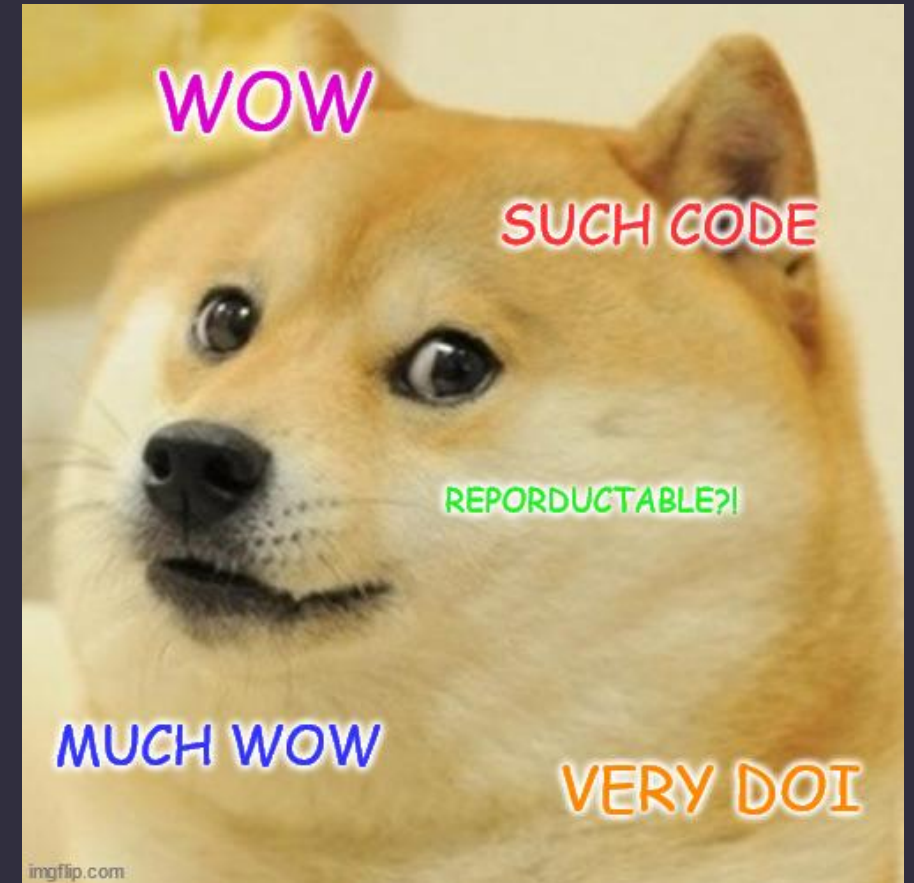
Further reading

❑ A beginner's guide to docs, by Eric Holscher (co-founder of readthedocs)

❑ Clean Architecture - A Craftmans guide to software structure and design, and, Clean Code - A Handbook of Agile Software Craftsmanship
  ▪ Two established textbooks by Robert C Martin

❑ The Single Responsibility Principle blog post by Robert C Martin
  ▪ And the entire Clean Code Blog by same author

❑ The DrWatson paper and references therein

❑ Short course on scientific computing and package development by Tim Holy

❑ Git information page on remotes

❑ A Guide to Reproducible Code, by the British Ecological Society

while the only thing that will make you a strong coder is commitment and hard work, having fun while doing it will speed up the process ☺
I wish your newfound POWERS will guide you to a better future, and when you are the group leaders of the future, you will appreciate code accordingly!

smoots smoots many kisses! 👄❤ 👄❤ 👄❤ 👄❤ 👄❤ 👄❤