Reproducible computing environments for Python and R

Bioinformatics task force workshop

24/03/2022

Matteo Tiberti
Nikola Tom

Bioinformatics task force

- Point of contact for people that do bioinformatics in the institute
- dcrc-bioinfo@cancer.dk
- Point of contact for HPC resources
- Organization of courses and workshops

Save the date

- Next workshops in 2022
  - **24th of May**: Send your computer, use their computer: workshop on containers technology
  - **29th of August**: keeping track of your code with git and GitHub
  - **17-18th of November**: Automating your work with the Snakemake pipeline engine

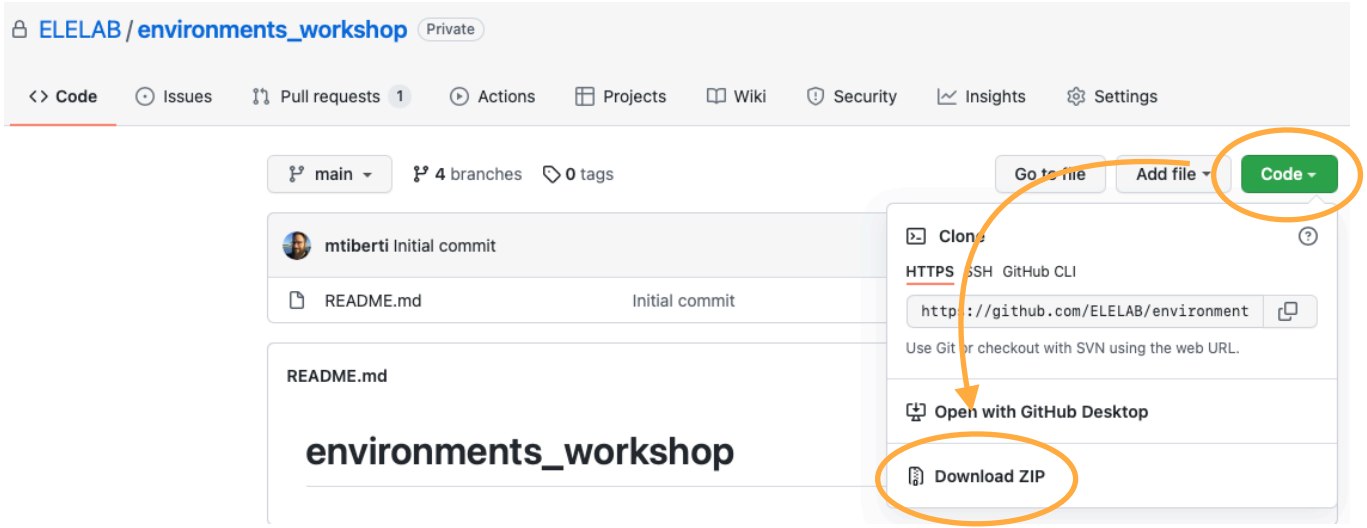
This workshop

- 10:00-10:20 Introduction
- 10:20-11:20 Python virtual environments
- 11:20-11:30 Break
- 11:30-12:15 R virtual environments with Renv
- 12:15-13:00 Lunch break
- 13:00-15:00 Conda environments

This workshop

Course material:

https://github.com/ELELAB/environments_workshop/



Replicability in computational science

Towards FAIR principles for research software

Cit

Issue title: FAIR Data, Systems and Analysis

Guest editors: Paul Groth and Michel Dumontier

Article type: Position Paper

Authors: Lamprecht, Anna-Lena[a; *] | Garcia, Leyla[b] | Kuzak, Mateusz[c; d] | Martinez, Carlos[e] | Arcila, Ricardo[f] | Martin Del Pico, Eva[g] | Dominguez Del Angel, Victoria[h] | van de Sandt, Stephanie[i] | Ison, Jon[j] | Martinez, Paula Andrea[k] | McQuilton, Peter[l] | Valencia, Alfonso[m; n] | Harrow, Jennifer[o] | Psomopoulos, Fotis[p] | Gelpi, Josep LL.[q; r] | Chue Hong, Neil[s; t] | Goble, Carole[u] | Capella-Gutierrez, Salvador[v; **]

Open Access | Published: 08 October 2020

Publish or perish, but do not forget your software artifacts

Robert Heumüller ✉, Sebastian Nielebock, Jacob Krüger & Frank Ortmeier

Empirical Software Engineering **25**, 4585–4616 (2020) | Cite this article

3168 Accesses | **7** Citations | **3** Altmetric | Metrics

PROTOCOLS article

Front. Neuroinform., 04 January 2018 | https://doi.org/10.3389/fninf.2017.00069

Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research

9 Pages • Posted: 8 Sep 2013

Victoria Stodden
University of Illinois at Urbana-Champaign - Graduate School of Library and Information Science

Sheila Miguez
Columbia University

Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions

Fabien C. Y. Benureau[1,2,3*] **and** **Nicolas P. Rougier**[1,2,3]

[1]INRIA Bordeaux Sud-Ouest, Talence, France
[2]Institut des Maladies Neurodégénératives, Université de Bordeaux, Centre National de la Recherche Scientifique UMR 5293, Bordeaux, France
[3]LaBRI, Université de Bordeaux, Bordeaux INP, Centre National de la Recherche Scientifique UMR 5800, Talence, France

Enhancing reproducibility for computational methods

VICTORIA STODDEN, MARCIA MCNUTT, DAVID H. BAILEY, EWA DEELMAN, YOLANDA GIL, BROOKS HANSON, MICHAEL A. HEROUX, JOHN P.A. IOANNIDIS, AND MICHELA TAUFER

Authors Info & Affiliations

SCIENCE • 9 Dec 2016 • Vol 354, Issue 6317 • pp. 1240-1241 • DOI: 10.1126/science.aah6168

Replicability in computational science

Towards FAIR principles for research software

Issue title: FAIR Data, Systems and Analysis

Guest editors: Paul Groth and Michel Dumontier

Article type: Position Paper

Authors: Lamprecht, Anna-Lena[a; *] | Garcia, Leyla[b] | Kuzak, Mateusz[c; d] | Martinez, Carlos[e] | Arcila, Ricardo[f] | Martin Del Pico, Eva[g] | Dominguez Del Angel, Victoria[h] | van de Sandt, Stephanie[i] | Ison, Jon[j] | Martinez, Paula Andrea[k] | McQuilton, Peter[l] | Valencia, Alfonso[m; n] | Harrow, Jennifer[o] | Psomopoulos, Fotis[p] | Gelpi, Josep Ll.[q; r] | Chue Hong, Neil[s; t] | Goble, Carole[u] | Capella-Gutierrez, Salvador[v; **]

Open Access | Published: 08 October 2020

Publish or perish, but do not forget your software artifacts

Robert Heumüller, Sebastian Nielebock, Jacob Krüger & Frank Ortmeier

Empirical Software Engineering **25**, 4585–4616 (2020) | Cite this article

3168 Accesses | **7** Citations | **3** Altmetric | Metrics

Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research

9 Pages • Posted: 8 Sep 2013

Victoria Stodden
University of Illinois at Urbana-Champaign - Graduate School of Library and Information Science

Sheila Miguez
Columbia University

PROTOCOLS article

Front. Neuroinform., 04 January 2018 | https://doi.org/10.3389/fninf.2017.00069

Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions

Fabien C. Y. Benureau[1,2,3*] **and**  **Nicolas P. Rougier**[1,2,3]

[1]INRIA Bordeaux Sud-Ouest, Talence, France
[2]Institut des Maladies Neurodégénératives, Université de Bordeaux, Centre National de la Recherche Scientifique UMR 5293, Bordeaux, France
[3]LaBRI, Université de Bordeaux, Bordeaux INP, Centre National de la Recherche Scientifique UMR 5800, Talence, France

Enhancing reproducibility for computational methods

VICTORIA STODDEN, MARCIA MCNUTT, DAVID H. BAILEY, EWA DEELMAN, YOLANDA GIL, BROOKS HANSON, MICHAEL A. HEROUX, JOHN P.A. IOANNIDIS, AND MICHELA TAUFER

Authors Info & Affiliations

SCIENCE • 9 Dec 2016 • Vol 354, Issue 6317 • pp. 1240-1241 • DOI: 10.1126/science.aah6168

Replicability in computational science

- You wrote some piece of code that does some analysis for a scientific work
- Your work is published and you made your code available
- What now?

Your code should be…

Re-runnable	You must be able to rerun your program
Repeatable	You obtain the same results as your former run(s), and they are correct
Reproducible	Another researcher can rerun your code and obtain the same results
Reusable	Your code is available and can be easily reused and modified, internally or externally
Replicable	Your code can be rewritten from the description in the paper and your results reproduced

Replicability in computational science

- You wrote some piece of code that does some analysis for a scientific work
- Your work is published and you made your code available
- What now?

Your code should be…

Re-runnable	You must be able to rerun your program
Repeatable	You obtain the same results as your former run(s), and they are correct
Reproducible	Another researcher can rerun your code and obtain the same results
Reusable	Your code is available and can be easily reused and modified, internally or externally
Replicable	Your code can be rewritten from the description in the paper and your results reproduced

Rerunnable code

```
import random

x = 0
walk = []
for i in xrange(10):
    step = random.choice([-1, +1])
    x += step
    print x,
```

Your code relies on many **moving parts**
(in space and time)

- Libraries/Packages (dependency trees)
- Interpreter (if any)
- Operating System
- Hardware

Ubuntu 18.04

```
teo@kb-bioinfo01:test-delta$ python random_walk.py
1 0 -1 0 1 2 3 2 3 2
```

Ubuntu 20.04

```
$ python random_walk.py
  File "random_walk.py", line 8
    print x,
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(x, end=" ")?
$
```

Repeatable code

```
>>>
>>> print(3 / 5)
0
>>>
```

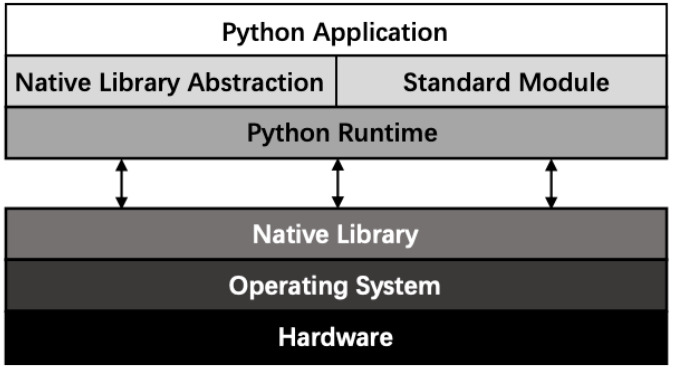
```
>>>
>>> print(3 / 5)
0.6
>>>
```

Repeatable code

Reproducibility

- Code doesn’t exist in a vacuum. Usually it depends first on the environment you develop it on (i.e. your computer)
- Giving someone access to your code is not enough to ensure that it is reproducible
- The *computing environment* your code runs it should come along with it

Computing environment



- A computing environment is the set of packages, libraries, interpreter, OS… that make running your program possible
- Each element of this list should be specified as well as possible (including **versioning**!)
- However…

Computing environment

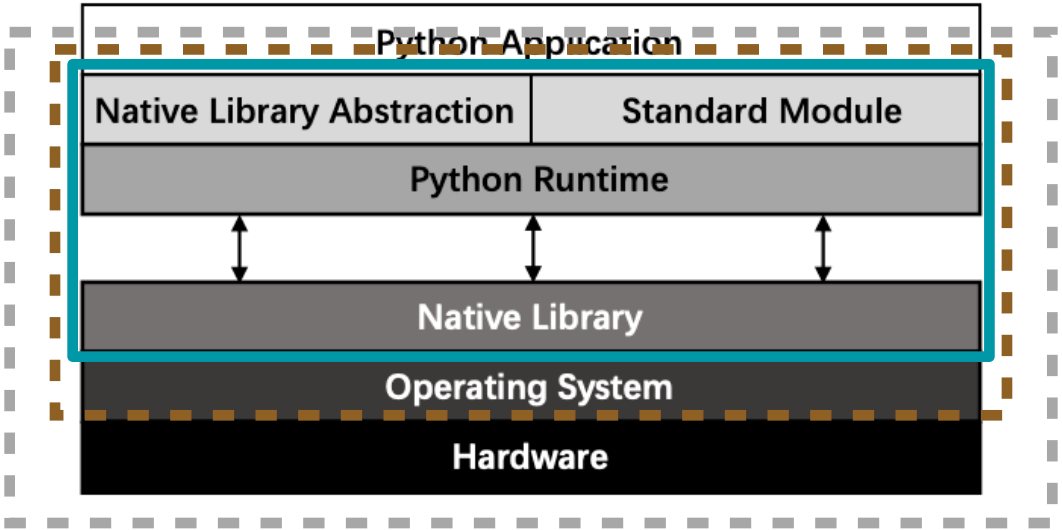
- Documenting your environment is a chore
    - Time-consuming and error-prone
    - Keeping track of it over time
    - Synchronizing different computers
- Reproducing someone else's environment is also time-expensive and error-prone
    - Environments are not very easy to share
- Different projects might require different environments

Computing environment

- Tools for package/dependency management and **environment managers**. They create a specific computer environment so that…
    - They install packages and their dependencies (**package management**)
    - They keep track of your dependencies and their versions (**tracking**)
    - They make the environment for a specific project **isolated**
    - They support writing the status of your environment to a file that is easy to **share**

Computing environment

- There isn't a "one size fits all" for all of them - each programming language has its own strategy (or more than one…)
- They depend on web resources
- They don't cover the whole stack
- They do cover most cases



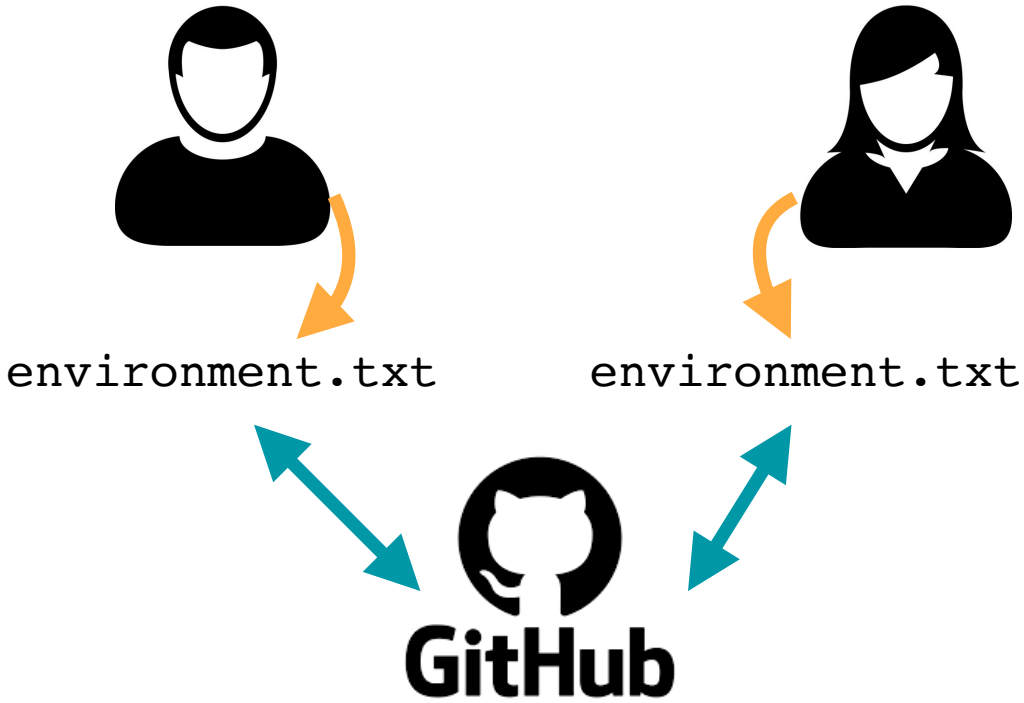
Computing environment workflow

There isn’t a “one size fits all” for all of them - each programming language has its own strategy (or more than one…). However they all follow a basic pattern of operation:

1. Creation of the env
2. Activation of your environment. This isolates your operations from the rest of the system
3. Installation of packages from an online source
4. Export the env as a text file.
   The file can be used for step 1

Computing environment workflow

- Environments can be shared (i.e. multiple people can use the same environment)
- However it can become complicated if several people share the same env
- A better strategy is to save your environment as a file and then share it together with your code, e.g. through git/GitHub

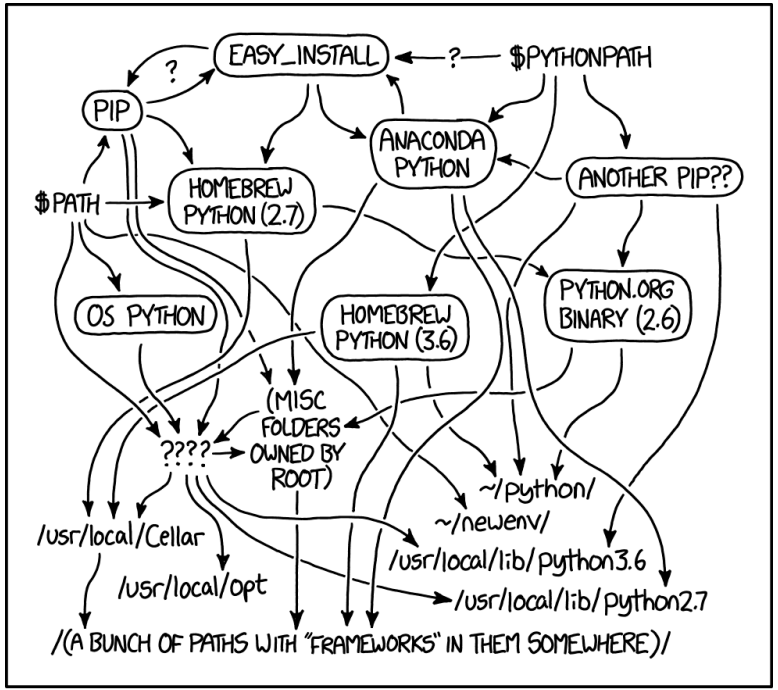


Introducing environments

	Python virtualenv	Renv	Conda
Language	Python	R	Python, R and more
Interface	Command line (Linux/Unix and Windows)	R prompt / Rstudio	Command line (Unix/ Linux), GUI (Windows)
The environment includes	Interpreter, packages	Packages	System libraries, interpreter, packages
Difficulty	Easy	Very easy	Easy to hard

Python virtual environments

- Python has 7+ options for virtual environments
- We're going to see the most used/popular one which is based on the **virtualenv** Python package
- In order to use it you need to have:
  - A Python installation (any version >= 2.7)
  - The **virtualenv** Python package installed
  - An internet connection



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

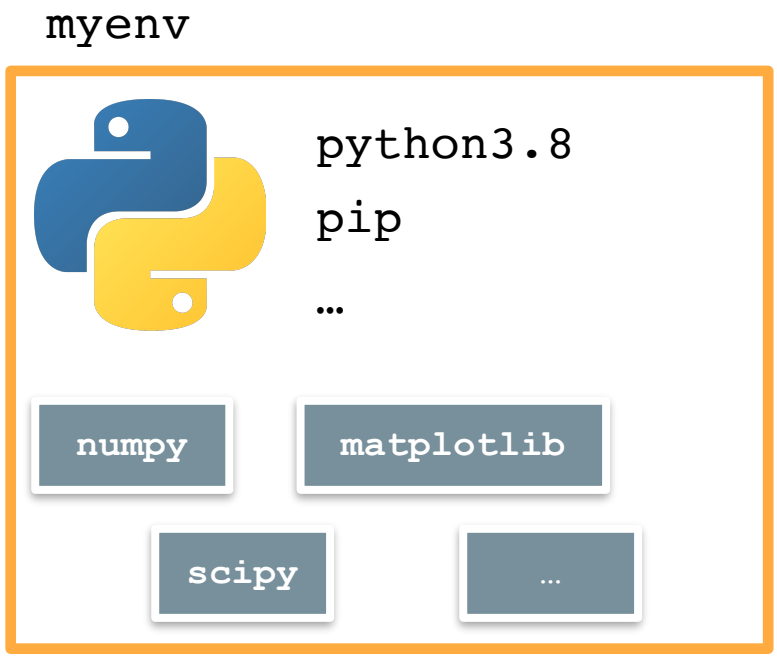
Python virtual environments

1. Create an environment (e.g. called “myenv”) as a self-contained directory.

2. Activate your environment

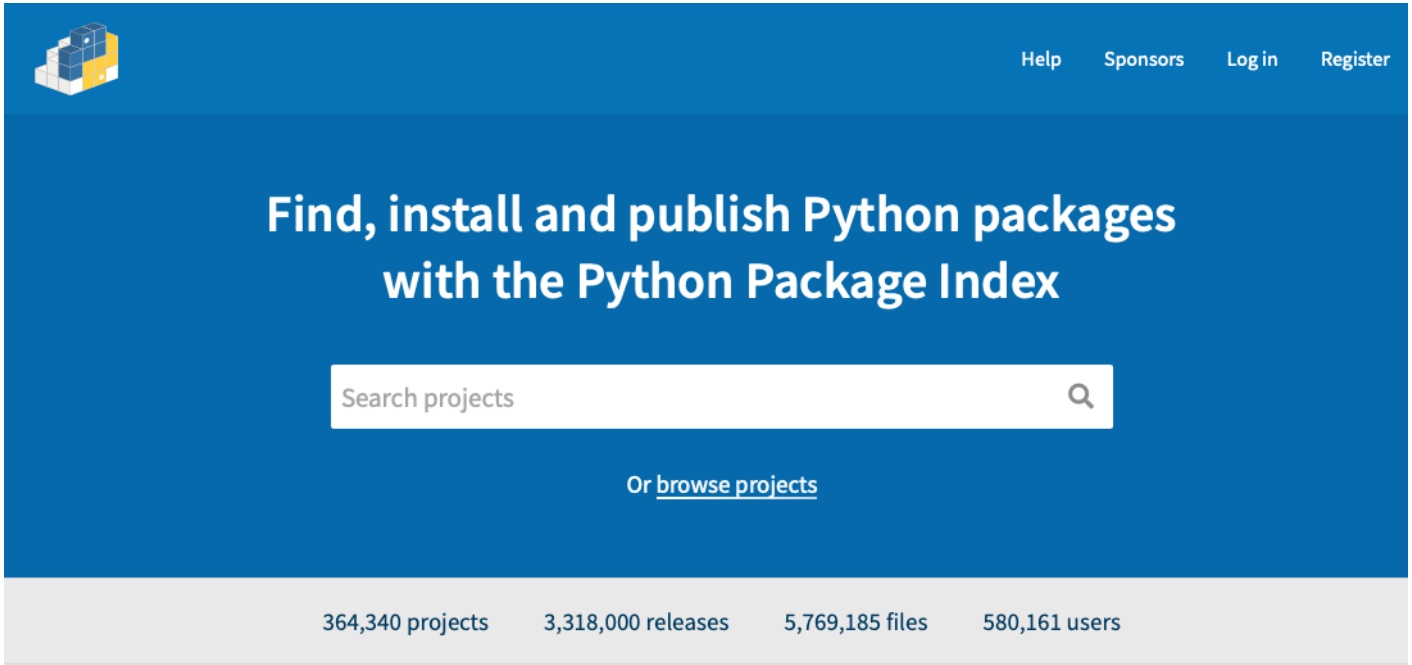
3. Install packages you need to your environment using **pip**

4. Export your environment as a file. It includes **all and only** the packages **you have installed and their dependencies**

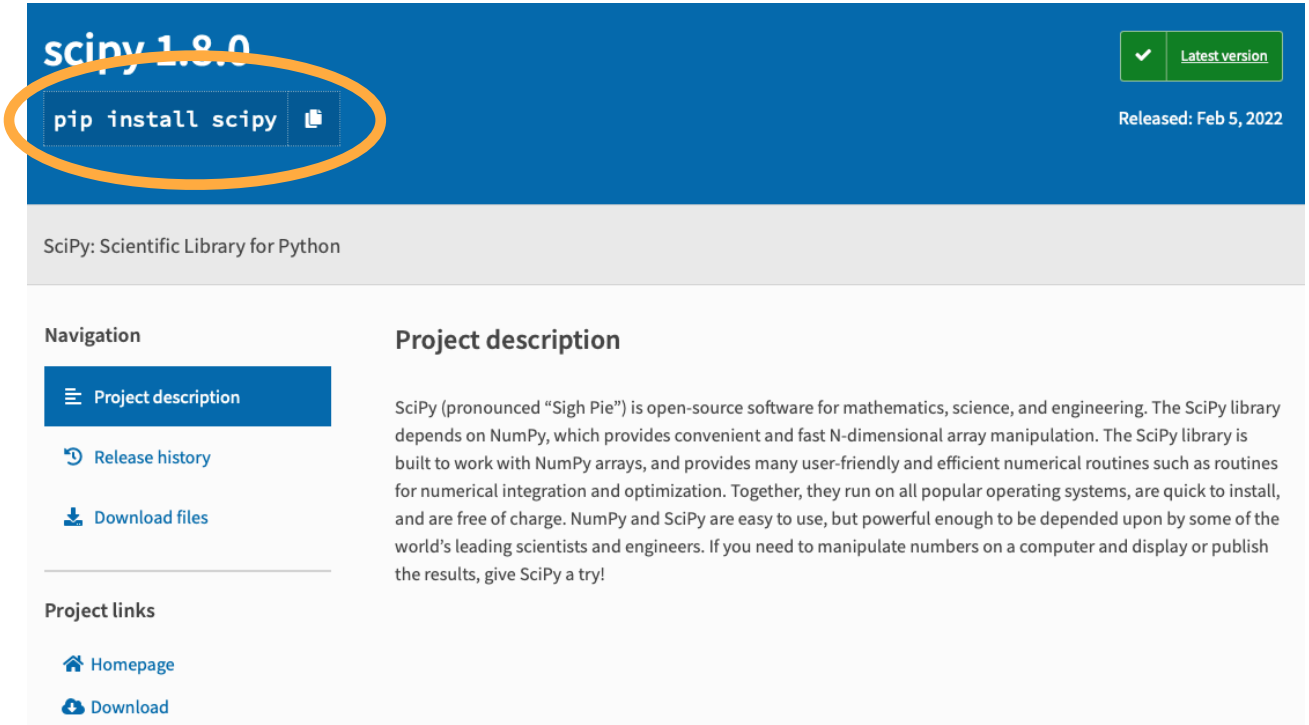


Python virtual environments

Python packages are downloaded from the **Python package index** (https://pypi.org)



Python virtual environments



Demonstration time

Course tutorial:
https://github.com/ELELAB/environments_workshop/blob/main/virtualenv/tutorial.md

Additional learning material:
https://virtualenv.pypa.io/en/latest/user_guide.html#

R virtual environments

- R also has different solutions for package management/environments
- We will see **Renv** which is an R package that handles environments
  https://rstudio.github.io/renv/articles/renv.html
- Differently from virtualenv:
  - It all happens **within R**
  - The environment has no control over the R version, just package versions

R virtual environments

1. Start working on your project. Include `library()` calls you need in your code

2. Create your environment. Renv identifies automatically what packages you need, installs them in the environment and saves the environment to a file

3. Continue working. Periodically save the state of your environment.

Demonstration time

Course tutorial:
https://github.com/ELELAB/environments_workshop/blob/main/renv/tutorial.md

Additional learning material:
https://cran.r-project.org/web/packages/renv/vignettes/renv.html