

Getting started

ar017 · 7 July 2026

demolab is a lab notebook for computational science. You write the science as code, an experiment runs it and emits data, a write-up reads that data, and one build turns the whole repository into a website and a set of PDFs. Numbers and figures come from the run, never retyped, so a result on the page can't quietly drift from the code that produced it. A coding agent operates it in plain language; you stay in the loop.

Install

Open a coding agent in an empty folder and paste:

Run `uvx demolab-cli init` here, then follow its GETTING-STARTED runbook strictly.

It runs the GETTING-STARTED runbook and sets everything up with you: the toolchain — `uv` for Python and `Typst` for publishing — then your own lab, born from the `demolab-cli` package, and your first experiment. You drive everything through the `demolab` command, which wraps `uv` and `typst`; you never call `pip` or `python` directly.

Your first run

The agent drives this: GETTING-STARTED stands the lab up, then builds your first experiment with you — your own science, or a suggested starter — and you watch it land on a live page. By hand, the loop is:

```
uvx demolab-cli init # lay down the lab: root files + writings/ experiments/ tools/
artifacts/
demolab dev          # serve the site at localhost:3000, live-reloading on save
uv run python experiments/exp000.py # run an experiment end to end (once you've
written one)
```

Change a parameter, run the experiment again, and watch the page update — figures and numbers, no prose touched. That's the whole point. `demolab build` compiles everything once; `demolab test` runs the suite. (Want a worked example to model your first experiment on? `demolab docs STARTERS` prints the reference dir — `monte-carlo-pi` is the canonical starter; build it as your own.)

How a lab is laid out

Four content directories, each with one job:

- `tools/`: the **science**. Reusable models and solvers, one directory per tool. A tool emits **data** (a `numbers.json` plus the data behind each figure), never plots, and stays language-agnostic.
- `experiments/`: the **runners**. One `expNNN.py` per experiment: it calls a tool (or computes inline), renders the figures into `artifacts/data/expNNN/`, and stages a `numbers.json` of the headline metrics.
- `writings/`: the **write-ups**. One `.typ` per entry. It reads the run, `#let run = json("/artifacts/data/expNNN/numbers.json")`, embeds the figures, and pulls every number from that file. Prose-only articles (like this one) and slide decks live here too.
- `artifacts/`: the committed **record**: `data/<id>/` (figures + numbers) and `pdfs/` (a PDF per entry, plus a bound book). The built website (`site/`) is regenerated by CI, so it's gitignored.

The engine itself lives in the installed `demolab-cli` package: a black box you never edit, updated with an ordinary dependency bump.

The loop

The discipline is one rule with teeth: *nothing on the page is typed by hand*.

1. A tool computes and writes data files.
2. A runner turns that data into figures and a `numbers.json`.
3. A write-up reads the `numbers.json` and embeds the figures.
4. `demolab build` compiles the repository into a website, a PDF per entry, and a book.

Change the code, rebuild, and the prose, figures, and numbers update together. Every result carries the git provenance of the run that made it.

Publish

One build emits three things from a single source: a **website** (HTML with real, selectable math), a **PDF per entry**, and a bound **book**. To put it online, `demolab deploy-setup` drops a GitHub Pages workflow; enable Pages in the repository settings and push. The site uses relative links, so it works under any path.

Driving it with an agent

`demolab` is meant to be run by a coding agent. You type a **name in capitals** and it acts:

- **HELP**: list everything it can do.
- A **guide** name (`RULES`, `HOUSESTYLE`, `SLIDES`, `STRUCTURE`, `GLOSSARY`, `SUPPORT`): it walks you through that reference.
- A **runbook** name (`LINT`, `DOCTOR`, `MIGRATE-STACK`, ...): it runs that procedure step by step.

The reference layers are documented right here in this collection: a page for each **guide** (always-on conventions), plus a **runbooks** index (the on-demand procedures).