

Writing a writeup

ar024 · 11 July 2026

A writeup is one `writings/<id>.typ` file: a `meta` block and a `body` block. Everything else is helpers from `lib.typ`, imported root-relative. This page is a working reference, organised by what you are trying to do. Its examples use a minimal experiment, `exp000`, that estimates π by Monte Carlo (drop N seeded points in the unit square, count the fraction inside the quarter circle). For how the prose should read, see [Writing style](#) this is the mechanics.

Start a new entry

Two top-level definitions make a writeup, and `build.py` discovers entries by them. The filename stem is the entry id: `exp000.typ` becomes `exp000`, served at `exp000.html` with a PDF at `pdfs/exp000.pdf`. `meta` carries `title`, `date`, and the optional `description`, `collection`, `status`, and `order`. `order` is an integer: any entry carrying one makes its collection list in that curated order (ascending) instead of newest-first, and unranked entries trail.

```
#let meta = (
  title: "Estimating  $\pi$  by Monte Carlo sampling",
  date: "2026-05-13",
  description: "One line, shown under the title and on listings.",
  collection: "estimators",
  status: "final",
  order: 2,
)

#let body = [ ... ]
```

Write headings normally with `==`. Each one gets a slug id on the web, so any section is deep-linkable (`exp000.html#methods`) with a permalink that appears on hover.

Pull in the run

Numbers and figures come from the run, never retyped. `data-file(rel)` resolves a path under `artifacts/data/`, and `json(...)` reads the run's `numbers.json` into a value you index. That object holds a `config` block (the run's args plus a `_provenance` stamp) and the headline metrics beside it, here `pi_estimate` and `abs_error`.

```
#import "../demolab/lib.typ": data-file, numbers-table
#let run = json(data-file("exp000/numbers.json"))
```

Interpolate a metric inline with `#calc.round(run.pi_estimate, digits: 4)`. To print the run's parameters and metrics as a table, pass it to `numbers-table`: it lists the `config` as parameters and the remaining keys as metrics, straight from the run.

```
#numbers-table(run, title: "Monte Carlo run parameters")
```

Place figures and video

A data figure is a standard Typst `#figure` wrapping an `#image` of a tool-rendered asset under the run's data directory. Give every image an `alt:` line so the web is accessible. The caption should stand on its own, and any number in it comes from the run too.

```
#figure(  
  image(data-file("exp000/scatter.png"), width: 100%, alt: "Sampled points  
  coloured by whether they fall inside the quarter circle"),  
  caption: [#calc.round(run.config.n, digits: 0) points in the unit square,  
  coloured by the  
    inside-circle test, giving  $\pi \approx$  #calc.round(run.pi_estimate, digits: 4).],  
)
```

A rendering plays on the web with `video`, referenced by `basename` (`build.py` emits every `mp4` as a bundle asset). In the PDF it becomes a short note pointing at the web edition.

```
#import "/.demolab/lib.typ": video  
#video("samples.mp4", caption: [Samples accumulating as the estimate  
converges on  $\pi$ .])
```

Write math

Write real Typst math: inline with `$...$`, display with a block. It renders as selectable MathML on the web and is typeset in the PDF, so never paste an equation as an image. Define every symbol after the equation.

```
$ \hat{\pi} = 4 \cdot 1/N \sum_{i=1}^N \mathbb{1}[x_i^2 + y_i^2 \leq 1] $
```

where $\hat{\pi}$ is the estimate, N the number of samples, $x_i, y_i \sim \text{Uniform}(0, 1)$ the coordinates of the i -th sample, and $\mathbb{1}[\cdot]$ the indicator that is 1 when the point falls inside the quarter circle and 0 otherwise. The estimate converges as $\hat{\pi} \approx \pi$ for large N .

Cite prior work

Two helpers manage numbering, linking, and the web hover-popover, so never hand-type a bracket or a list. `#cite(1, 2)` renders [1, 2] and links each number to its entry; attach it directly to the word it follows, with no space. `#reference-list(items)` renders the numbered References section, each item a dict with free-Typst `text` and an optional `doi` that links to `https://doi.org/<doi>`. Numbering is positional: keep the inline numbers in step with the list order.

```
#import "/.demolab/lib.typ": cite, reference-list

The estimator's error falls as  $1/\sqrt{N}$ , the standard Monte Carlo
rate#cite(1).

#reference-list((
  (text: [Author A, Author B (2020). Title. _Journal_ 12:34.], doi: "10.1000/
xyz"),
))
```

Mark work in progress

Set `status` to a lifecycle value: `draft`, `building`, `revising`, or `final`. `final` (the default) shows nothing, so only unfinished work is flagged; the badge appears next to the date everywhere the entry lists. When a figure's asset is not ready yet, `pending-figure` reserves its footprint so the page does not reflow when the real plot lands, and it still numbers as a normal "Figure N".

```
#import "/.demolab/lib.typ": pending-figure
#pending-figure(
  caption: [Absolute error against sample count N.],
  note: [re-run in flight],
  ratio: 16 / 9,
)
```

Swap it back to a real `#figure` once the asset exists: a `pending-figure` left in a `final` entry is a lint smell.

Stamp provenance

Close the body with the git-commit footer. Pass the run's `config`, which carries the `_provenance` block the runner stamped: `provenance-footer` prints the short commit and the build date, so a reader can trace the page back to the exact code that made it.

```
#import "/.demolab/lib.typ": provenance-footer
#provenance-footer(run.config)
```

That is the whole authoring surface. For the prose, math, figure, and caption conventions that keep the lab reading as one voice, see [Writing style](#).