

Using another language

ar026 · 11 July 2026

demolab ships Python: the skeleton lays down a uv environment, plotting helpers, and a provenance stamp, and every worked reference is Python. But Python is the opinionated example, not a requirement. The contract that connects a tool to an experiment is file-based and language-neutral, so your science can stay in whatever language it already lives in.

Why any language works

A tool is never imported — it is reached by **running its command line** (a subprocess), and it communicates only through files: `config.json`, `output.json`, `manifest.json`, a `run.sh` reproducer, and its figure data (`.csv`, `.json`, `.npz` — your choice of format). Any executable that parses arguments and writes those files is a valid tool. The write-up reads `numbers.json`, and Typst typesets from that plus the rendered figures — neither cares what produced them. So the only language-bound layer is the tool itself. See The contract for the full boundary.

Two ways in

Pick the smallest change that gets your language into the lab.

1. **Hybrid — tools in your language, Python stays the glue.** Write `tools/<tool>/tool.<ext>` in MATLAB, Julia, R, or another language; the runner still shells out to it and keeps a minimal uv environment purely to stage figures and plot. Least churn, and the recommended path — you rewrite only the science.
2. **Full switch — tool and runner in your language.** Rewrite the runner too, so it renders its own figures and writes `numbers.json`, then run that runner on the new runtime and drop the Python dependencies. Julia is the natural fit here: it can own the tool, the runner, and the plotting. Take this only if you want Python out entirely.

What never changes

Either path leaves the rest of demolab untouched: the engine (Typst, inside the `demolab-cli` package), your `writings/*.typ` (they read `numbers.json` exactly as before), `artifacts/`, and the `numbers.json` schema. Provenance still holds — you reimplement the stamp (git commit, dirty flag, UTC timestamp) in the new language’s run-setup helper.

What your language needs

Three primitives, which every mainstream scientific language has: run non-interactively from a command line, write JSON for the contract files, and write figure data (a CSV is enough) — plus a unit-test runner. MATLAB (`matlab -batch`), Octave, Julia (`JSON3, CSV`), and R (`jsonlite, write.csv`) all qualify.

Ask your agent

Say *MIGRATE-STACK* (or “use Julia instead of Python”) and your agent follows the *MIGRATE-STACK* runbook: it confirms the runtime, ports the tool contract into your language, wires the runner entry point and `demolab test` to it, and offers the hybrid path first.