Lean for the Working Mathematician

Dr. Stefano Rocca

Soft Skills activities

Timetable: 12 hrs, Torre Archimede, Room 2BC30:

Dec 2, 2026	14:30-16:30
Dec 3, 2026	14:30-16:30
Dec 9, 2026	14:30-16:30
Dec 11, 2026	14:30-16:30
Dec 16, 2026	14:30-16:30
Dec 18, 2026	14:30-16:30

See also the Calendar of Activities on https://dottorato.math.unipd.it/calendar

See next pages for Course requirements, Aim, Outline of the course

Lean for the Working Mathematician

Hands-on course in reading, navigating, and writing formal mathematics

Duration: 12 hours

Format: Live coding, guided exercises, code reading, and individual or paired practice, with a small mini-project connected with participants' mathematical interests if time permits. There will be no final assessment.

Course requirements

No previous experience with Lean, programming, or formal verification is required. Familiarity with Git/GitHub could be useful but is not mandatory. The course is intended primarily for mathematicians, although participants from other fields are also welcome: the vast majority of toy examples use undergraduate-level mathematics. Participants should bring a laptop on which they are permitted to install applications and which can run Visual Studio Code with the Lean extension. We will take our time going through the installation process together, and a browser-based environment can be used while installation is under way.

Aim

The course is meant to help researchers get started with formalized mathematics and become comfortable using Lean4 in practice. Participants will learn how to read theorem statements and proof states, inspect definitions and hypotheses, find useful results in Mathlib, make informed use of automation, and take the first steps towards a formalization of their own. A recurring theme will be the difference between the theorem one has in mind, the statement written in Lean, and the proof object ultimately checked by the kernel. Formalization often brings hidden assumptions, ambiguous formulations, edge cases, and important choices of definition into view.

Outline of the course

The course will touch on the following topics, not necessarily in the order listed. The selection and emphasis may be adapted to the interests of the participants.

- **Working environment and first interaction.** Installation of Lean4 and a version-pinned Mathlib project; use of Visual Studio Code, the Infoview, and a browser editor for short experiments. First declarations with `#check`, `def`, `example`, `lemma`, and `theorem`, together with the interpretation of errors, warnings, and incomplete declarations.
- **Statements, specifications, and mathematical intent.** Comparison of informal claims with their formal counterparts, including examples in which Lean checks a correct proof of an unintended statement. Quantifiers, hidden hypotheses, exceptional cases, and the responsibility of the formalizer for choosing a faithful theorem statement.
- **Proof states and elementary proof construction.** The local context, target, goals, and subgoals. Core logical forms and tactics such as `intro`, `exact`, `apply`, `constructor`, `cases`, `rcases`, `use`, `refine`, and `have`, with continual comparison to the corresponding mathematical reasoning.
- **Mathematical language and abstraction.** Functions, predicates, relations, sets, images, preimages, and extensionality. Definitions, namespaces, overloaded notation, implicit arguments, structures, typeclasses, coercions, and type inference as they occur in ordinary mathematical code.
- **Equality, rewriting, and structured arguments.** Definitional and propositional equality, computation and `rfl`, unfolding, rewriting, and simplification through `unfold`, `rw`, and `simp`. Development of readable multi-step arguments using intermediate facts and `calc` blocks.
- **Reading and navigating Mathlib.** How to approach an unfamiliar source file through imports, sections, namespaces, variables, declarations, instances, and nearby lemmas. Use of hover information, completion, go to definition, source inspection, `#print`, `#synth`, and `#find` to reconstruct the mathematical interface of a development.

- **Searching for definitions and theorems.** Use of the MATHLIB API documentation, LeanSearch, Moogle, and Loogle. Local search and suggestion commands such as `exact?`, `apply?`, `rw?`, and `simp?`, with emphasis on comparing candidate results and understanding their assumptions.
- **Automation and proof engineering.** Controlled simplification and domain-specific tactics including `norm_num`, `ring`, `linarith`, `nlinarith`, and `omega`, together with proof search using `aesop` and `grind`. Comparison of explicit, library-driven, and automated proofs in terms of readability, robustness, explanatory value, and maintainability.
- **Small-scale independent formalization.** Selection and scoping of a manageable mathematical example, reuse of existing definitions, decomposition into intermediate lemmas, use of scratch files and temporary `sorry`s. Refactoring, documentation, and reproducibility. The aim is to leave participants able to continue experimenting with a topic connected to their own interests.
- **AI-assisted and large-scale formalization.** A concluding discussion of how AI and increasingly automated methods are being used in larger formalization projects, including theorem proving, library search, proof repair, and the coordination of extended developments. We will discuss the possibilities opened by these methods, their current limitations, and the questions they raise for mathematical practice. Mathematicians who choose to formalize may need to strengthen complementary skills: specifying the intended theorem correctly; choosing suitable definitions and abstractions; decomposing a project into meaningful lemmas; checking that formal statements match mathematical intent; reviewing assumptions, axioms, imports, and dependencies; and deciding whether the resulting code is readable, reusable, and maintainable.

Teaching format

The course will be taught primarily through live coding, with a continuous back-and-forth between informal mathematical reasoning and its formal expression in Lean. Theoretical ideas will be introduced through concrete examples and immediately explored in practice. At least half of the course time will be devoted to hands-on work, including guided exercises, code reading, and direct experimentation in Lean.

References and resources

1. K. Buzzard, *Formalising Mathematics*. Course notes and exercises for mathematicians using Lean and MATHLIB.
2. J. Avigad and P. Massot, *Mathematics in Lean*. A systematic introduction to mathematical formalization with Lean and MATHLIB.
3. P. Massot, *A Glimpse of Lean*. A short, exercise-driven introduction designed for mathematical audiences.
4. J. Avigad, L. de Moura, S. Kong, S. Ullrich, and contributors, *Theorem Proving in LEAN4*. Reference material on logic, proof terms, tactics, structures, and typeclasses.
5. F. van Doorn and M. Rothgang, *Bonn Lean Course, WiSe 2025/26*. Executable lecture files, assignments, installation guidance, and project material.
6. MATHLIB API documentation, the tactic reference, and the search services LeanSearch, Moogle, and Loogle.