Lean: Introduction

What is Lean?

A programming language that checks mathematical proofs.

```
theorem exists_prime_ge (n : ℕ) :
    ∃ p, n ≤ p ∧ Nat.Prime p := by
  exact Nat.exists_infinite_primes n
```

Mathlib

- A library of mathematical **definitions** and **theorems**, written in Lean.
- Written and maintained by **humans**. Every addition is reviewed.
- Very high quality but slow progress: only a small % of math is covered.

E.g. even *stating* (not proving) the main theorems of recent *Annals* papers is often blocked by missing definitions. The Annals Challenge (Buzzard et al.) has stated the main theorem from 50 papers from the 2020s, out of about 230.

Reading a Lean statement

```
theorem exists_deriv_eq_slope                    -- name
    (f : ℝ → ℝ) {a b : ℝ}                        -- variables
    (hab : a < b)                                -- hypotheses
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b,                           -- conclusion
      deriv f c = (f b - f a) / (b - a)
```

Let $f\colon \mathbb{R} \to \mathbb{R}$ and $a < b$. If f is continuous on $[a, b]$ and differentiable on (a, b), then there is $c \in (a, b)$ with $f'(c) = \frac{f(b)-f(a)}{b-a}$.

Reading a Lean statement: the name

```
theorem exists_deriv_eq_slope
    (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

- Every theorem has the shape

  theorem *name hypotheses* **:** *conclusion* **:=** *proof*
- `:=` reads “is proved by”.
- The name is just a label, read in pieces:
  `exists_deriv_eq_slope` = “there *exists* a point where the *derivative equals* the *slope*”.
- Mathlib names follow conventions, so you can often guess them.

Reading a Lean statement: the variables

```
theorem exists_deriv_eq_slope
    (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

- `x : T` means "x is an element of T" (Lean says: *x has type T*).
- `ℝ → ℝ` is the set of functions from $\mathbb{R}$ to $\mathbb{R}$. So `(f : ℝ → ℝ)` means "let $f\colon \mathbb{R} \to \mathbb{R}$".
- `{a b : ℝ}` means "let $a, b \in \mathbb{R}$".
- Why `( )` for f but `{ }` for a, b? Coming up.

Reading a Lean statement: hypotheses

```
theorem exists_deriv_eq_slope
    (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

- `(hab : a < b)` means “assume $a < b$, and call this assumption `hab`”.
- Hypotheses are written just like variables: a name, a colon, and what it is.
- `a < b` is a *statement*; `hab` is a *proof* of it.
- Hypothesis names usually start with `h`. `hab` = “hypothesis about a, b”.

Reading a Lean statement: more hypotheses

```
theorem exists_deriv_eq_slope
    (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

- Functions are applied without brackets: `f x` means $f(x)$.
  So `ContinuousOn f S` means "f is continuous on S".
- Brackets only group: `ContinuousOn f (Set.Icc a b)`.
- Intervals: `Set.Icc a b` $= [a, b]$, `Set.Ioo a b` $= (a, b)$
  `I` = interval, `c` = closed, `o` = open (also `Ico`, `Ioc`).
- `DifferentiableOn ℝ f S`: "f is differentiable on S" (over $\mathbb{R}$).

All of these are **Mathlib definitions**, and so is $\mathbb{R}$ itself.

Reading a Lean statement: the conclusion

```
theorem exists_deriv_eq_slope
    (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

- The last colon `:` separates the hypotheses from the conclusion.
- `∃ c ∈ S, P` means "there is $c \in S$ such that P".
- `deriv f c` means $f'(c)$.
- `(f b - f a) / (b - a)` means $\dfrac{f(b) - f(a)}{b - a}$.

Applying a theorem: ( ) vs { }

```
theorem exists_deriv_eq_slope
    (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfd : DifferentiableOn ℝ f (Set.Ioo a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

Using a theorem looks like applying a function to arguments:

```
exists_deriv_eq_slope f hab hfc hfd
```

- `( )`: you write the argument (here `f`, `hab`, `hfc`, `hfd`).
- `{ }`: Lean fills it in. No `a`, `b` above: `hab : a < b` already says what they are.
- When *reading* a statement, both just mean “let”.

Applying a theorem: an example

The mean value theorem for $f(x) = x^2$ on $[0, 1]$:

```
example : ∃ c ∈ Set.Ioo (0 : ℝ) 1,
    deriv (fun x : ℝ ↦ x ^ 2) c = (1 ^ 2 - 0 ^ 2) / (1 - 0) := by
  have hab : (0 : ℝ) < 1 := by norm_num
  have hfc : ContinuousOn (fun x : ℝ ↦ x ^ 2) (Set.Icc 0 1) := by
    fun_prop
  have hfd : DifferentiableOn ℝ (fun x : ℝ ↦ x ^ 2) (Set.Ioo 0 1) := by
    fun_prop
  exact exists_deriv_eq_slope (fun x : ℝ ↦ x ^ 2) hab hfc hfd
```

- `hab` is a proof that $0 < 1$ (`by norm_num`: "by computation").
- `hfc`, `hfd`: proofs that x^2 is continuous and differentiable (`by fun_prop`: "by standard facts").
- From `hab : 0 < 1`, Lean sees that $a = 0$ and $b = 1$. That is why `a`, `b` are in `{ }`.

Proofs: the goal

In VS Code, the **infoview** shows what we have, and what is left to prove (after ⊢).

Infoview: before the last line

```
hab : 0 < 1
hfc : ContinuousOn (fun x ↦ x ^ 2) (Set.Icc 0 1)
hfd : DifferentiableOn ℝ (fun x ↦ x ^ 2) (Set.Ioo 0 1)
⊢ ∃ c ∈ Set.Ioo 0 1, deriv (fun x ↦ x ^ 2) c = (1 ^ 2 - 0 ^ 2) / (1 - 0)
```

```
  exact exists_deriv_eq_slope (fun x : ℝ ↦ x ^ 2) hab hfc hfd
```

Infoview: after

```
No goals
```

`exact` *p*: "*p* proves exactly the goal." Here *p* is the mean value theorem, applied to x^2 with `hab`, `hfc`, `hfd`.

Quiz: true or false?

```
theorem quiz_1 (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

Quiz: true or false?

```
theorem quiz_1 (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b)) :
    ∃ c ∈ Set.Ioo a b, deriv f c = (f b - f a) / (b - a)
```

False. This is the mean value theorem without differentiability. Take $f(x) = |x|$ on $[-1, 2]$. The slope is $\frac{2-1}{3} = \frac{1}{3}$, but `deriv f c` is always -1, 1, or 0.

Quiz: true or false?

```
theorem rolle (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfab : f a = f b) :
    ∃ c ∈ Set.Ioo a b, deriv f c = 0
```

Quiz: true or false?

```
theorem rolle (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfab : f a = f b) :
    ∃ c ∈ Set.Ioo a b, deriv f c = 0
```

True, but misformalized.

- Take $f(x) = |x|$ on $[-1, 1]$. It gives $c = 0$, but $|x|$ is not differentiable at 0.
- Where f is not differentiable, Mathlib defines `deriv f c` to be 0. So `deriv f c = 0` does not mean "$f'(c) = 0$".

Quiz: true or false?

```
theorem rolle (f : ℝ → ℝ) {a b : ℝ}
    (hab : a < b)
    (hfc : ContinuousOn f (Set.Icc a b))
    (hfab : f a = f b) :
    ∃ c ∈ Set.Ioo a b, deriv f c = 0
```

True, but misformalized.

- Take $f(x) = |x|$ on $[-1, 1]$. It gives $c = 0$, but $|x|$ is not differentiable at 0.
- Where f is not differentiable, Mathlib defines `deriv f c` to be 0. So `deriv f c = 0` does not mean "$f'(c) = 0$".

By the way: Mathlib includes this as a true theorem, called `exists_deriv_eq_zero`.

Quiz: which can Lean prove?

```
theorem no_square_root_of_two : ¬ ∃ x, x ^ 2 = 2

theorem square_root_of_two : ∃ x, x ^ 2 = 2
```

Quiz: which can Lean prove?

```
theorem no_square_root_of_two : ¬ ∃ x, x ^ 2 = 2

theorem square_root_of_two : ∃ x, x ^ 2 = 2
```

Only the first! No type was given, so Lean picks the default: `x : ℕ`.
(In VS Code, hover over `x` to see this.)

Quiz: which can Lean prove?

```
theorem no_square_root_of_two : ¬ ∃ x, x ^ 2 = 2

theorem square_root_of_two : ∃ x, x ^ 2 = 2
```

Only the first! No type was given, so Lean picks the default: `x : ℕ`.
(In VS Code, hover over `x` to see this.)

Stated correctly, over $\mathbb{R}$:

```
theorem square_root_of_two : ∃ x : ℝ, x ^ 2 = 2
```

Now this one is provable (take $x = \sqrt{2}$), and the first one is false.

Quiz: true or false?

```
theorem harmonic_series : ∑' n : ℕ, 1 / ((n : ℝ) + 1) = 0
```

`∑' n : ℕ` is the infinite sum $\sum_{n=0}^{\infty}$.

Quiz: true or false?

```
theorem harmonic_series : ∑' n : ℕ, 1 / ((n : ℝ) + 1) = 0
```

`∑' n : ℕ` is the infinite sum $\sum_{n=0}^{\infty}$.

True! Lean proves it.

- In Mathlib, the sum of a series that does not converge is *defined* to be 0.
- So `∑' n, a n = S` does not say that the series converges.

Quiz: true or false?

```
theorem harmonic_series : ∑' n : ℕ, 1 / ((n : ℝ) + 1) = 0
```

`∑' n : ℕ` is the infinite sum $\sum_{n=0}^{\infty}$.

True! Lean proves it.

- In Mathlib, the sum of a series that does not converge is *defined* to be 0.
- So `∑' n, a n = S` does not say that the series converges.

Same for integrals: Lean also proves

```
theorem integral_inv_zero_one : ∫ x in (0 : ℝ)..1, 1 / x = 0
```

Quiz: true or false?

```
-- n is perfect if it is the sum of its divisors
def IsPerfect (n : ℕ) : Prop := ∑ d ∈ n.divisors, d = n

theorem finitely_many_perfect : {n : ℕ | IsPerfect n}.Finite
```

`∑ d ∈ n.divisors, d` means $\sum_{d|n} d$: the index before the comma, the summand after it.

Quiz: true or false?

```
-- n is perfect if it is the sum of its divisors
def IsPerfect (n : ℕ) : Prop := ∑ d ∈ n.divisors, d = n

theorem finitely_many_perfect : {n : ℕ | IsPerfect n}.Finite
```

`∑ d ∈ n.divisors, d` means $\sum_{d|n} d$: the index before the comma, the summand after it.

True! Lean proves it. Did we just solve an open problem?

Quiz: true or false?

```
-- n is perfect if it is the sum of its divisors
def IsPerfect (n : ℕ) : Prop := ∑ d ∈ n.divisors, d = n

theorem finitely_many_perfect : {n : ℕ | IsPerfect n}.Finite
```

`∑ d ∈ n.divisors, d` means $\sum_{d \mid n} d$: the index before the comma, the summand after it.

True! Lean proves it. Did we just solve an open problem?

- No: `n.divisors` includes *n* itself. `Nat.divisors 6 = {1, 2, 3, 6}`.
- So the sum is at least $n + 1$ for $n \geq 2$, and only 0 and 1 are "perfect".
- The definition is wrong. Mathlib's `Nat.Perfect` uses `n.properDivisors`.

Misformalization: Erdős problem #480

Erdős: for any $x_1, x_2, \ldots \in [0,1]$, are there infinitely many m, n with $|x_{m+n} - x_m| \leq \frac{1}{\sqrt{5}\,n}$? (Yes: Chung–Graham.)

```
theorem erdos_480 : ∀ (x : ℕ → ℝ), (∀ n, x n ∈ Set.Icc 0 1) →
    {(m, n) | (m) (n) (_ : m ≠ 0)
              (_ : |x (m + n) - x m| ≤ 1 / (√5 * n))}.Infinite
```

`{(m, n) | (m) (n) (_ : P) (_ : Q)}` is the set of pairs (m, n) with P and Q. The `_` is a hypothesis with no name: like `(hab : a < b)`, but we never refer to it.

Source: Google DeepMind's Formal Conjectures, 2025.

Misformalization: Erdős problem #480

Erdős: for any $x_1, x_2, \ldots \in [0,1]$, are there infinitely many m, n with $|x_{m+n} - x_m| \leq \frac{1}{\sqrt{5}\,n}$? (Yes: Chung–Graham.)

```
theorem erdos_480 : ∀ (x : ℕ → ℝ), (∀ n, x n ∈ Set.Icc 0 1) →
    {(m, n) | (m) (n) (_ : m ≠ 0)
              (_ : |x (m + n) - x m| ≤ 1 / (√5 * n))}.Infinite
```

`{(m, n) | (m) (n) (_ : P) (_ : Q)}` is the set of pairs (m, n) with P and Q. The `_` is a hypothesis with no name: like `(hab : a < b)`, but we never refer to it.
Source: Google DeepMind's Formal Conjectures, 2025.

- The statement says `m ≠ 0`. It should say `n ≠ 0`.
- Take $n = 0$: then $|x_m - x_m| = 0$ and $\frac{1}{\sqrt{5}\cdot 0} = 0$ in Lean. So $0 \leq 0$, for every m.
- Nov 2025: an AI (Harmonic's Aristotle) found this two-line "proof". Fixed by making n a positive integer.

Misformalization: the smooth Poincaré conjecture

Smooth Poincaré: a smooth n-manifold homotopy equivalent to S^n is diffeomorphic to S^n. Open for $n = 4$.

```
def ConjectureFor (n : ℕ) : Prop :=
  ∀ (M : Type) [TopologicalSpace M] [T2Space M] [ChartedSpace (ℝ^n) M],
    M ≃ₕ 𝕊^n → Nonempty (M ≃ₜ 𝕊^n)
def SmoothConjectureFor (n : ℕ) : Prop :=
  ∀ (M : Type) [TopologicalSpace M] [ChartedSpace (ℝ^n) M] [IsManifold (𝓡 n) ∞ M],
    M ≃ₕ 𝕊^n → Nonempty (M ≃ₘ⟮𝓡 n, 𝓡 n⟯ 𝕊^n)
theorem smooth_dimension_four : SmoothConjectureFor 4    -- "open"
```

`ConjectureFor n` is a name for the statement "Poincaré holds in dimension n" (`: Prop` = it is a statement). `≃ₕ`, `≃ₜ`, `≃ₘ`: homotopy equivalent, homeomorphic, diffeomorphic. `[T2Space M]`: M is Hausdorff. Source: Formal Conjectures, until Sept 2026.

Misformalization: the smooth Poincaré conjecture

Smooth Poincaré: a smooth n-manifold homotopy equivalent to S^n is diffeomorphic to S^n. Open for $n = 4$.

```
def ConjectureFor (n : ℕ) : Prop :=
  ∀ (M : Type) [TopologicalSpace M] [T2Space M] [ChartedSpace (ℝ^n) M],
    M ≃ₕ 𝕊^n → Nonempty (M ≃ₜ 𝕊^n)
def SmoothConjectureFor (n : ℕ) : Prop :=
  ∀ (M : Type) [TopologicalSpace M] [ChartedSpace (ℝ^n) M] [IsManifold (𝓡 n) ∞ M],
    M ≃ₕ 𝕊^n → Nonempty (M ≃ₘ⟮𝓡 n, 𝓡 n⟯ 𝕊^n)
theorem smooth_dimension_four : SmoothConjectureFor 4    -- "open"
```

`ConjectureFor n` is a name for the statement "Poincaré holds in dimension n" (`: Prop` = it is a statement). `≃ₕ`, `≃ₜ`, `≃ₘ`: homotopy equivalent, homeomorphic, diffeomorphic. `[T2Space M]`: M is Hausdorff. Source: Formal Conjectures, until Sept 2026.

- The smooth version forgot `[T2Space M]`. Mathlib manifolds need not be Hausdorff.
- Counterexample, like the line with two origins: S^n with a second copy of $\{x_0 < 1\}$ glued along $\{x_0 < 0\}$.
- So it was false for every $n \geq 1$, including the "open" case $n = 4$.

Misformalization: a false axiom

A 2026 project formalizing a complex structure on S^6 assumed some classical theorems as **axioms**:

```
/-- Cellular chains of a CW complex are naturally quasi-isomorphic
    to singular chains. Reference: [Hat02, Theorem 2.35]. -/
axiom integralCWCellularChainModel
    (Y : Type) [TopologicalSpace Y] [Topology.CWComplex (Set.univ : Set Y)] :
    IntegralCWCellularChainModel Y
```

`axiom`: a statement Lean accepts without proof. In words: for every CW complex Y, cellular homology computes singular homology. Source: github.com/deancureton/sphere-six-complex, issue #151, Aug 2026.

Misformalization: a false axiom

A 2026 project formalizing a complex structure on S^6 assumed some classical theorems as **axioms**:

```
/-- Cellular chains of a CW complex are naturally quasi-isomorphic
    to singular chains. Reference: [Hat02, Theorem 2.35]. -/
axiom integralCWCellularChainModel
    (Y : Type) [TopologicalSpace Y] [Topology.CWComplex (Set.univ : Set Y)] :
    IntegralCWCellularChainModel Y
```

`axiom`: a statement Lean accepts without proof. In words: for every CW complex Y, cellular homology computes singular homology. Source: github.com/deancureton/sphere-six-complex, issue #151, Aug 2026.

- Mathlib's `CWComplex` does not assume Hausdorff (Hatcher's CW complexes are Hausdorff).
- Two points with the indiscrete topology form a Mathlib CW complex with two 0-cells: cellular $H_0 = \mathbb{Z}^2$. But the space is path-connected: singular $H_0 = \mathbb{Z}$.
- So the axiom is false, Lean can derive `False`, and *every* theorem in the project was vacuous. Fixed by adding `[T2Space Y]`.

Autoformalization: the paper



Chodosh–Gianocca, *No compromise in the liquid drop model* (arXiv 2608.11517):

Theorem. For $\Omega \subset \mathbb{R}^3$ let $\mathcal{E}(\Omega) = P(\Omega) + \frac{1}{2}\int_\Omega \int_\Omega \frac{dx\,dy}{|x-y|}$, where P is the perimeter, and let $V_* = \frac{5(2-2^{2/3})}{2^{2/3}-1} \approx 3.51$.

- If $0 < V \leq V_*$, the minimizers of $\mathcal{E}$ among sets of volume V are exactly the balls.
- If $V > V_*$, there is no minimizer.

Autoformalization: the statement



First formalize the *statement* in `Showcase.lean`, with `sorry` as the proof. Used for development.

```
theorem main (V : ℝ) (hV : 0 < V) :
    (V ≤ criticalVolume →
      ∀ Ω : Set AmbientSpace, IsFixedVolumeMinimizer V Ω ↔ IsBallUpToNull V Ω) ∧
    (criticalVolume < V → ¬ ∃ Ω : Set AmbientSpace, IsFixedVolumeMinimizer V Ω)
    := by sorry
```

- About 180 lines: definitions (perimeter, Coulomb energy, minimizer), main theorems, and sanity checks.
- This is the file a human must read carefully: misformalizations live here.

Autoformalization: how `Showcase.lean` is written




```
abbrev AmbientSpace := EuclideanSpace ℝ (Fin 3)
def perimeter (Ω : Set AmbientSpace) : ℝ≥0∞ := ⨆ (X : AmbientSpace → AmbientSpace) …
def coulombEnergy (Ω : Set AmbientSpace) : ℝ≥0∞ :=
  (2 : ℝ≥0∞)⁻¹ * ∫⁻ p in Ω ×ˢ Ω, (ENNReal.ofReal ‖p.1 - p.2‖)⁻¹
def energy (Ω : Set AmbientSpace) : ℝ≥0∞ := perimeter Ω + coulombEnergy Ω
def criticalVolume : ℝ := 5 * (2 - (2 : ℝ) ^ ((2 : ℝ) / 3)) / ((2 : ℝ) ^ ((2 : ℝ) / 3) - 1)
def IsFixedVolumeMinimizer (V : ℝ) (Ω : Set AmbientSpace) : Prop :=
  MeasurableSet Ω ∧ volume Ω = ENNReal.ofReal V ∧ perimeter Ω < ∞ ∧
    ∀ F : Set AmbientSpace, MeasurableSet F → volume F = ENNReal.ofReal V → energy Ω ≤ energy F
```

- Anything not in Mathlib is defined explicitly: `perimeter`, `coulombEnergy`, `criticalVolume`.
- Properties central to the theorem get their own name: `IsFixedVolumeMinimizer`.
- Abbreviations for readability: `AmbientSpace` $= \mathbb{R}^3$.
- Ideally, each definition/theorem can be matched to the paper.

Autoformalization: setting up a Lean project



In Codex or Claude Code, in an empty folder containing `paper.pdf`:

> *Set up this folder as a Lean project that depends on the latest Mathlib. Check that `lake build` works.*

- `lake` is Lean's build tool. The file `lakefile.toml` says the project needs Mathlib.
- `lake build` checks every Lean file in the project. No errors = every proof is correct.
- `lake exe cache get` downloads Mathlib precompiled (otherwise: hours).

```
name = "myproject"

[[require]]
name = "mathlib"
scope = "leanprover-community"
rev = "v4.35.0-rc3"

[[lean_lib]]
name = "MyProject"
```

lakefile.toml

Autoformalization: writing `Showcase.lean` with an agent



Then, in the same folder:

> *Write `Showcase.lean`: the main theorems of `paper.pdf`, with `sorry` as proofs, importing only Mathlib. Define explicitly anything not in Mathlib, and give key properties their own definitions. Match definitions/theorems to the paper as appropriate. Add sanity checks showing the definitions are not vacuous.*

- **We check**: read every line against the paper.
- **An agent checks**: ask a fresh agent to compare `Showcase.lean` with the paper and list every mismatch.

Autoformalization: `sorry`




```
theorem main : 2 + 2 = 5 := by
  sorry

#print axioms main
```

Lean's output

```
warning: declaration uses `sorry`
'main' depends on axioms: [sorryAx]
```

- `sorry` means "proof omitted". The file still compiles, even for $2 + 2 = 5$.
- But it is not a proof: Lean warns, and `#print axioms` shows `sorryAx`.
- `Showcase.lean` is all statements with `sorry`. The rest of the work: remove every `sorry`.

Autoformalization: the blueprint



A LaTeX document: the proof split into small lemmas (excerpt):

```
\begin{lemma}[The ABP auxiliary Neumann problem]\label{lem:abp-neumann}
\uses{prop:weak-neumann,thm:boundary-neumann}
Let $G\subset\R^3$ be a bounded connected smooth domain.  There is a smooth
$z:\overline G\to\R$ with $\Delta z=\Per(G)/|G|$ in $G$, $\partial_\nu z=1$ on $\partial G$.
\end{lemma}
```

- Proves everything needed that is not in Mathlib (here: BV functions, geometric measure theory, elliptic PDE).
- Each lemma is small enough to be one task for an agent.
- `\uses{...}` gives a dependency graph (the `leanblueprint` tool).
- NoCompromise: 41 chapters, 335 statements, 177 pages. Written by AI agents from the paper; a human approved every change to a statement.

Autoformalization: writing the blueprint with an agent



In Codex or Claude Code, in the project folder (with `paper.pdf` and `Showcase.lean`):

> `/goal` *Write a blueprint in* `blueprint/` *(leanblueprint format) for* `paper.pdf`*, proving every theorem in* `Showcase.lean`*. Split each proof into lemmas small enough to formalize one at a time, with* `\uses{...}` *for dependencies. No axioms: the only dependency is Mathlib, so prove anything Mathlib is missing. Done when every statement in* `Showcase.lean` *follows from Mathlib with no gaps.*

- `/goal` keeps the agent working until the “done when” condition holds.
- Read every statement it changes. Ask a second agent to audit the blueprint.

Autoformalization: agents prove the lemmas



`/goal` *Formalize* `blueprint/` *in Lean, one lemma at a time, with the same statements. No* `sorry`*, no axioms. Run* `lake build` *after every change. If a blueprint statement looks wrong, stop and ask me. Done when every theorem in* `Showcase.lean` *is proved.*

NoCompromise, in practice:

- 4 weeks, one agent: about 118,000 lines.
- 2 days, an orchestrating agent splitting the remaining chapters among many agents, each in its own copy of the project: done.
- Total: 216,000 lines of Lean in 1,206 files. Mathematical doubts went back to the user, not to the agents.

Autoformalization: the comparator



Compares `Showcase.lean` (trusted: we read it) with `Solution.lean` (the proofs, by agents). Checks that each listed theorem

- has exactly the same statement as in `Showcase.lean`,
- uses no other axioms (so no `sorry`),
- is accepted by the Lean kernel.

Catches an agent that changed a statement or definition, or added an axiom.

```
{ "challenge_module": "Showcase",
  "solution_module": "Solution",
  "theorem_names":
    ["NoCompromise.main", ...],
  "permitted_axioms":
    ["propext", "Classical.choice",
     "Quot.sound"] }
```

configuration (NoCompromise, shortened)

Built into Lean as `lake comparator`. Online: “Can I Trust This Proof?” at `live.lean-lang.org`.

Autoformalization: Palomar



A public registry of Lean-verified mathematics: `palomar-registry.org` (since Aug 2026, about 600 results). Each entry pins one GitHub commit and must pass:

1. **Comparator**, with the proof also checked by a second, independent kernel.
2. **A language-model review**: is `Showcase.lean` a fair rendering of the paper's claim?
3. **Disclosure** (`formalization.yaml`): authors, sources, AI use, limitations.

Not certified: that the Lean statement matches the paper, novelty, importance. "Palomar is not a journal."

Autoformalization: who checks what



- **Humans read**: the paper and `Showcase.lean`.
- The blueprint is only an intermediate step between the paper and Lean. Nobody needs to check it: if it is wrong, the Lean proof does not go through.
- **Machines check** everything else: for NoCompromise, 216,000 lines nobody has to read.
- So misformalizations are a human problem: they live in `Showcase.lean`.

Installing Lean

1. Install **VS Code**.
2. Install the **Lean 4** extension. It installs Lean for you.
3. Make a project with Mathlib:

```
lake new myproject math   # makes the project
cd myproject
lake exe cache get        # downloads Mathlib, precompiled
```

No install needed to try it: `live.lean-lang.org`

Lean is a programming language: running untrusted Lean code (from the internet, or from an agent) on your computer can be risky.

Demo

`Demo.lean` in VS Code

1. **Reading.** Ctrl-click `exists_deriv_eq_slope` and `ContinuousOn` to see them in Mathlib.
2. **The example.** Click through `hab`, `hfc`, `hfd` in the infoview. Delete the `exact` line, see the goal, type it back.
3. **Break it.**
   - delete `: ℝ` in `hfc`: x becomes a natural number;
   - delete `hfd`: the mean value theorem needs it;
   - `sorry` and `#print axioms`;
   - a false axiom proves $2 + 2 = 5$.
4. **A real project.** NoCompromise's `Challenge.lean` and its Palomar check, on GitHub.

Learning Lean

Learn

- **Natural Number Game**: build $\mathbb{N}$ from scratch, in the browser.
  `adam.math.hhu.de`
- **Lean Zulip**: ask questions, people are friendly.
  `leanprover.zulipchat.com`

Have an AI agent quiz you

- "Show me 5 Mathlib statements. I translate them to English, you check."
- "Write a Lean statement with a subtle misformalization. I find the bug."
- "Give me a theorem with `sorry`. Hint one step at a time."

Agents make mistakes too: check their Lean code compiles.