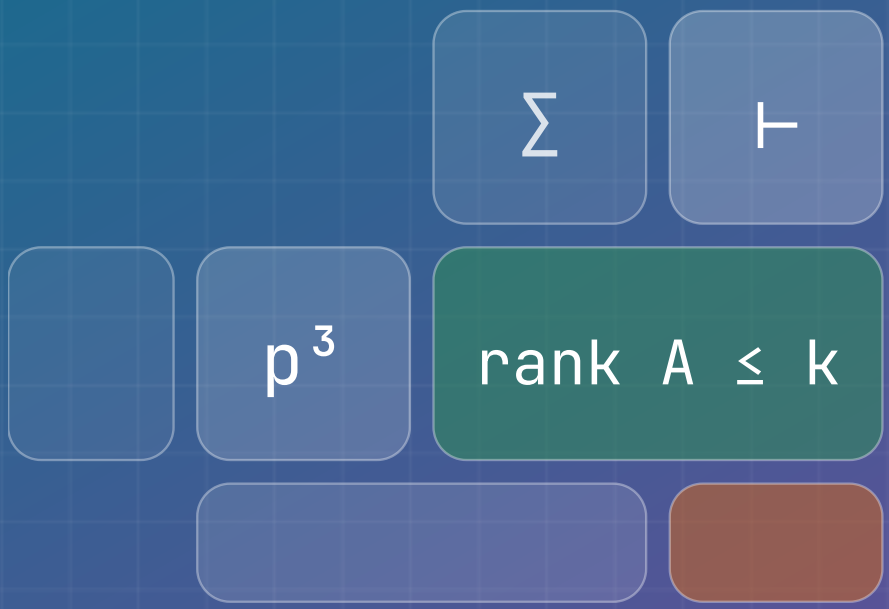
PRACTICAL GUIDE · LEAN 4

The Reusable layer: a practical guide

Machine-checked theorems you can use in your own Lean 4 proofs — from installing them to using every one of them.

Watch the video Simulations (Spanish)

v0.15.0 · 82 theorems Lean 4 Mathlib machine-checked open source



🏛️ https://karlesmarin.github.io/blueprints/reusable/ 💻 github.com/karlesmarin/formalizations @ v0.15.0

Contents

01	🎯	What you will be able to do	3
02	🧱	What the Reusable layer is	4
03	🧰	Before you start	6
04	🚀	Installing the layer	7
05	✍️	Your first theorem, word by word	10
06	🔎	Finding the theorem you need	11
07	🛠️	All the ways of using a theorem	13
08	🧪	Six recipes	14
09	🧭	Practical examples, start to finish	20
10	📚	Every theorem of the layer, by area	24
11	🩺	When something goes wrong	30
12	📦	Versions, updates, citing and contact	31
13	🗂️	Quick reference	32

🧩 **Recipes** — problem, theorem, code, what Lean checked, something to try.

🔗 **Links** — every theorem name links to its card on the showcase, always up to date.

💻 **Code** — every Lean example is compiled against the release.

🗂️ **Chapter 13** — a one-page quick reference to keep at hand.

01 CHAPTER 1 🎯 What you will be able to do

By the end of this guide you will have a Lean project that uses the Reusable layer, and you will know how to find, read, apply and combine any of its theorems. You do not need to read their proofs: Lean has checked each one, and you use its statement the way you use a theorem from a textbook.

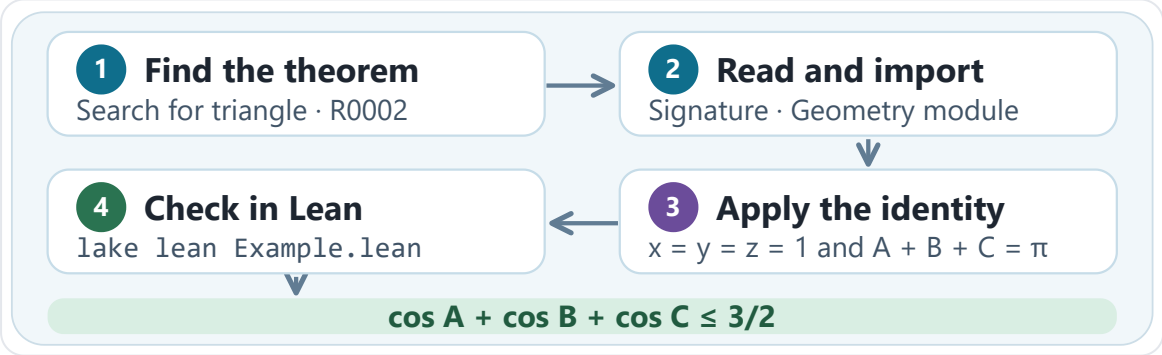
The layer is useful when Mathlib does not (yet) have the statement you need, or has it in a form that is awkward to apply. Three things it does for you, all worked out in this guide:

- 🔢 **Count real roots** of a polynomial from a few sign changes, without computing the roots (Recipe 2).
- ✅ **Certify that a quadratic form is a sum of k squares**: two theorems turn a check of signs into the decomposition (Recipe 4).
- 📐 **Prove a classical inequality in two lines** once the right identity is available (Recipe 1).

> 💡 **IDEA**
> **The whole method in one sentence.** Find the theorem, give it its arguments, prove its hypotheses, and take the part of its conclusion you need. Everything else in this guide is that sentence in detail.

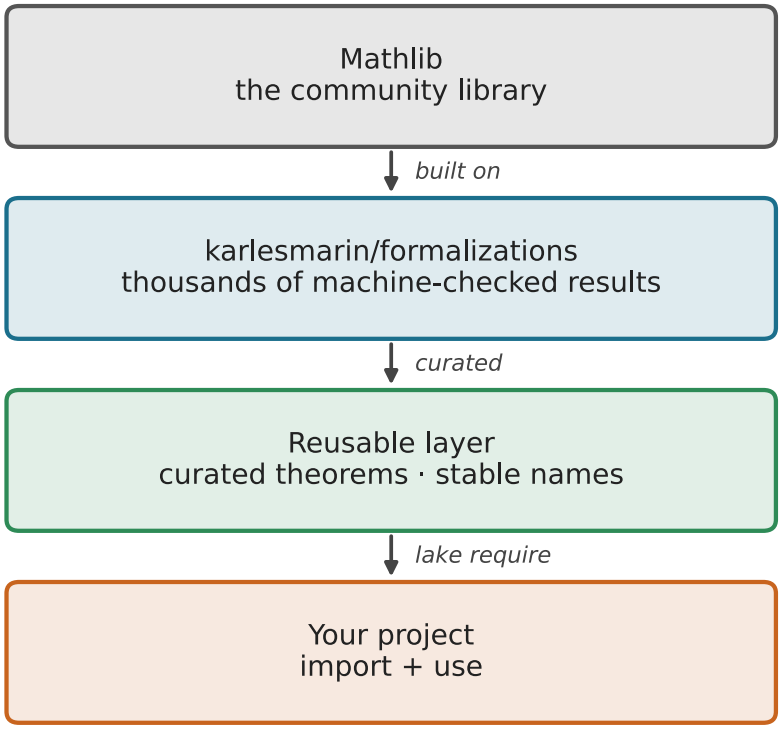
From a showcase card to a Lean proof

Can you prove **cos A + cos B + cos C ≤ 3/2** for the angles of a triangle? This route takes you from a search to the proof.



1. **Find the theorem.** Search for `triangle` in the showcase and open Wolstenholme's identity (card R0002).
2. **Read the signature and copy the import.** Start your file with `import Formalizations.Reusable.Geometry` and `open Formalizations.Reusable`. In the signature, `h` is the proof you supply that A + B + C = π.
3. **Apply the identity.** Recipe 1 uses the two squares to prove the general inequality. In Example 2, supply x = y = z = 1 and `h`. The identity makes 3 − 2(cos A + cos B + cos C) a sum of two squares.
4. **Check the proof.** Paste Example 2 into `Example.lean`. The proof uses `sq_nonneg` for each square and `nlinarith` to derive the bound. Run `lake lean Example.lean` in your configured project. Lean accepts the proof if the command exits successfully.

02 CHAPTER 2 🧱 What the Reusable layer is



The repository `karlesmarin/formalizations` holds thousands of machine-checked results, most of them written for my research papers. The **Reusable layer** is a small, curated part of it: theorems that are useful outside the paper they came from, under stable names of the form `Formalizations.Reusable.<name>`. Release `v0.15.0`, the one this edition describes, has 82 of them in 9 areas.

Every entry of the layer:

- is a **theorem proved in the archive**: the layer re-exports it, no proof is copied or rewritten;
- depends only on Lean's **standard axioms** (`propext` , `Classical.choice` , `Quot.sound`), checked in every release;
- has a **role**: 🔦 `API` , a lemma meant to be used inside other proofs, or 🏛️ `showcase` , a classical theorem, complete in itself, more often consulted than imported;
- has a **card** in the showcase: the statement in words, the exact Lean signature, the import it needs, a compiled example, where it comes from and which axioms it uses.

What it is **not**: it does not replace Mathlib (it is built on Mathlib and pins one Mathlib commit), and it is not a collection of tactics. It is a set of theorems.

📌 NOTE

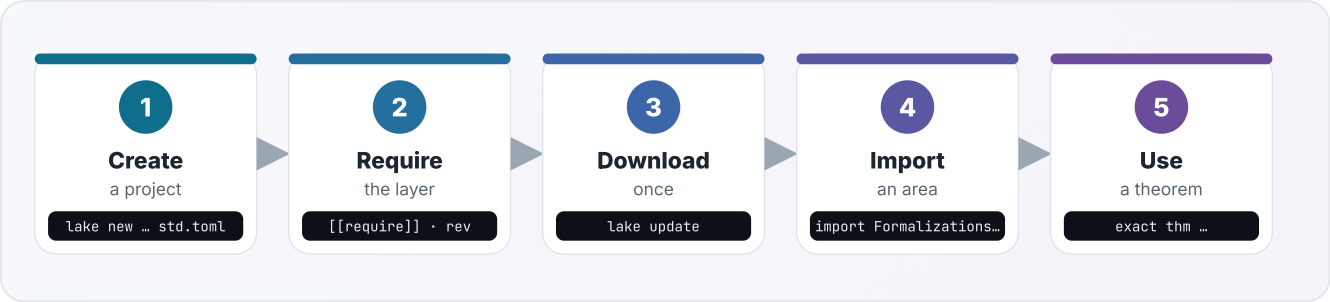
Stability. Names under `Formalizations.Reusable` are not removed within a major series. If a statement has to change incompatibly, the old name stays as a deprecated alias for two releases.

03 CHAPTER 3 Before you start

You need:

- **Lean 4 with** `elan`, the Lean version manager. If you have never installed Lean, follow https://lean-lang.org/install/; it takes a few minutes and installs VS Code support as well.
- **git**.
- About **5 GB of disk**: Mathlib, already compiled, is downloaded once.
- **Basic Lean**: what `theorem`, `example`, `by`, `exact` and `have` mean. If these words are new, the first chapters of *Mathematics in Lean* (https://leanprover-community.github.io/mathematics_in_lean/) are the best start; come back here after them.

04 CHAPTER 4 Installing the layer



4.1 In a new project, step by step

0 **Create the project.**

TERMINAL

```
lake +leanprover/lean4:v4.35.0 new myproject std.toml
cd myproject
```

> ⚠ **WATCH OUT**
> Use the **standard** template (`std.toml`), not `math` . The `math` template adds its own Mathlib, pinned to a different commit from the one this layer is built with, and you would end up with two Mathlibs that disagree. Mathlib comes into your project through this layer.

1 **Declare the dependency.** Open `lakefile.toml` and add at the end:

LAKEFILE.TOML

```
[[require]]
name = "formalizations"
git = "https://github.com/karlesmarin/formalizations"
rev = "v0.15.0"
```

`rev` pins the release: your project keeps working the same way until you change it.

📌 NOTE

Where the values come from. This edition of the guide is generated from release `v0.15.0` : its tag goes in `rev` , and its Lean version (`leanprover/lean4:v4.35.0` , the file `lean-toolchain` at the tag) in the `lake new` line. For a later release, take its tag from https://github.com/karlesmarin/formalizations/releases and the `lean-toolchain` at that tag; the showcase always names both.

2 **Download.**

TERMINAL

```
lake update
```

This fetches the layer and Mathlib, and downloads Mathlib already compiled. The first time takes a while (several gigabytes); later updates are quick.

3 **A first file.** Create `Example.lean` in the project folder:

LEAN 4

```
import Formalizations.Reusable.NumberTheory

open Formalizations.Reusable

#check wolstenholme_choose_modEq
```

4 **Check it.**

TERMINAL

```
lake lean Example.lean
```

Lean prints the statement of the theorem: your project is ready. On Linux and Windows Lake downloads the layer already compiled; on macOS it compiles it once.

⚠️ WATCH OUT

Avoid a bare `lake build` in a project from the standard template: its default target is an executable, and building it compiles every imported Mathlib module to native code, which takes hours. `lake lean <file>` is what you want; if your files belong to a library of your project, `lake build <library>` does the same.

4.2 In a project you already have

If your project uses **the same Lean and Mathlib** (the ones named in `lean-toolchain` and `lake-manifest.json` of the repository at the tag you pin; for `v0.15.0`, Lean `v4.35.0` and Mathlib commit `f0b4c81`), add the `[[require]]` block of step 1 to its `lakefile.toml` (or the equivalent `require` line in a `lakefile.lean`) and run `lake update`. If it uses another Mathlib, Lake will report a conflict: either move your project to these versions or wait for a release of the layer on yours.

4.3 Working in VS Code

Open the project folder in VS Code with the Lean 4 extension installed. When you open `Example.lean`, the **infoview** panel shows, at the cursor, the goal still to prove and the type of what is under the cursor. Hovering over a name such as `wolstenholme_choose_modEq` shows its statement; Ctrl-click (Cmd-click) jumps to where it is defined. This is the fastest way to read a theorem while you use it.

05 CHAPTER 5 ✍️ Your first theorem, word by word

Replace the `#check` line of `Example.lean` with:

LEAN 4

```
example : Nat.choose 13 6 ≡ 1 [MOD 7 ^ 3] :=
  (wolstenholme_choose_modEq 7 (by norm_num) (by norm_num)).1
```

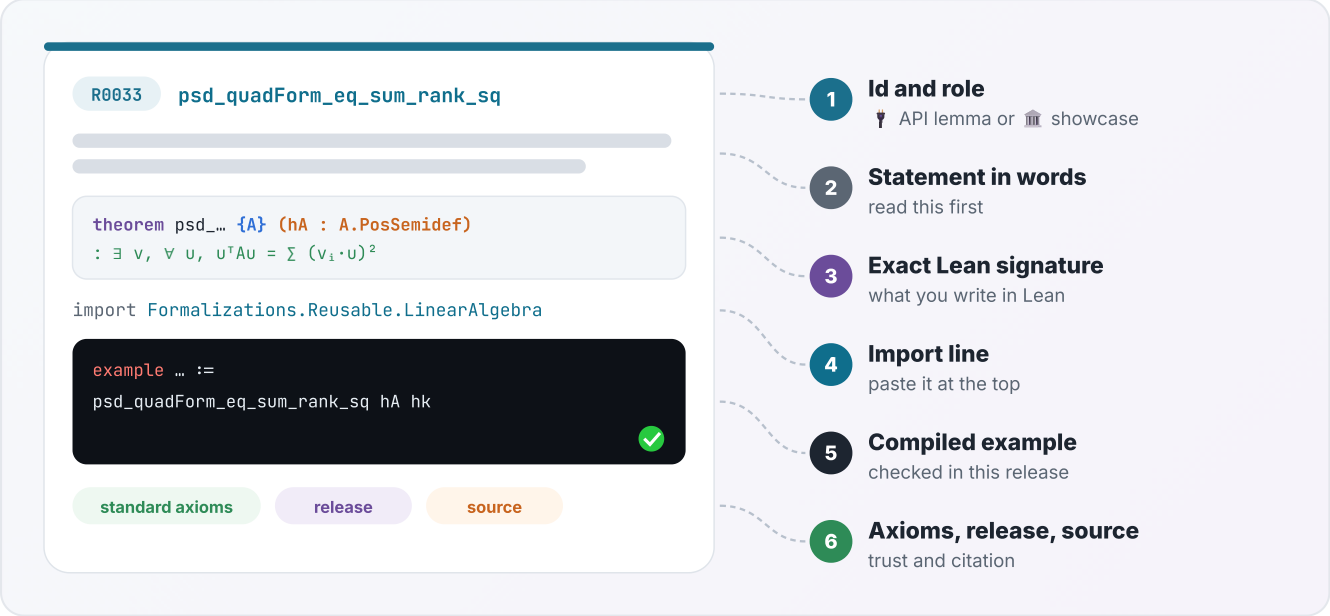
Run `lake lean Example.lean` again. No output means Lean accepted the proof. Piece by piece:

- `example : …` states something without naming it. Here: the binomial coefficient C(13, 6) = 1716 is congruent to 1 modulo $7^3 = 343$.
- `wolstenholme_choose_modEq` is the theorem of the layer (Wolstenholme, 1862): *for a prime $p \geq 5$, $C(2p - 1, p - 1) \equiv 1 \pmod{p^3}$ and $C(2p, p) \equiv 2 \pmod{p^3}$.*
- `7` is the value of p. Lean then asks for the two hypotheses of the theorem: that 7 is prime and that $5 \leq 7$. Each `(by norm_num)` proves one of them by computation.
- The theorem gives two congruences joined by "and"; `.1` takes the first, which is exactly our statement ($2 \cdot 7 - 1 = 13$ and $7 - 1 = 6$).

06 CHAPTER 6 🔎 Finding the theorem you need

The showcase (https://karlesmarin.github.io/blueprints/reusable/) lists every theorem with a search box. Search in words ("rank", "real roots", "triangle", "congruence") or browse by area. Each card shows:

- the statement in words and the exact **Lean signature**;
- the **import**: the module you write after `import` (the area of the theorem, for example `Formalizations.Reusable.LinearAlgebra`); `import Formalizations.Reusable` imports every area at once;
- a **compiled example**: code Lean checked in this release, ready to paste;
- the **axioms** it uses, the **release** (with its Lean and Mathlib versions), and the **paper** or classical source it comes from.



The manifest `Formalizations/Reusable/manifest.toml` holds the same information as data, one entry per theorem, if you prefer to search it with your own scripts. **Section 10** of this guide lists every theorem by area.

Inside Lean you can ask directly:

LEAN 4

```
#check psd_quadForm_eq_sum_rank_sq        -- its statement
#print axioms psd_quadForm_eq_sum_rank_sq -- which axioms it rests on
```

Reading a signature

LEAN · R0033 · FORMALIZATIONS.REUSABLE.LINEARALGEBRA

```
theorem psd_quadForm_eq_sum_rank_sq {n : Type*} [Fintype n] [DecidableEq n] {A : Matrix n n ℝ}
    (hA : A.PosSemidef) {k : ℕ} (hk : A.rank = k) :
    ∃ v : Fin k → (n → ℝ), ∀ u, u ⬝ᵥ (A *ᵥ u) = ∑ i, (v i ⬝ᵥ u) ^ 2
```

{ } implicit: Lean deduces it [ ] instance: Lean finds it ( ) you supply it ▇ the conclusion: what you get

- Arguments in **braces** `{n}`, `{A}`, `{k}` are *implicit*: Lean deduces them from the others; you do not write them.
- Arguments in **square brackets** `[Fintype n]` are *instances*: Lean finds them itself for the usual types (`Fin 3`, `ℕ`, `ℝ`, ...).
- Arguments in **parentheses** `(hA : …)`, `(hk : …)` are the ones you supply: here, a proof that `A` is positive semidefinite and a proof that its rank is `k`.
- What follows the last colon is the **conclusion**: what you get.

07 CHAPTER 7 All the ways of using a theorem

The same theorem can enter a proof in several ways. Here they are with one theorem, `matrix_rank_add_le` (the rank of a sum is at most the sum of the ranks).

TACTIC	USE IT WHEN	EXAMPLE
`exact`	the goal is exactly the conclusion	`exact matrix_rank_add_le A B`
`have`	you want the conclusion as a fact for later	`have h := matrix_rank_add_le A A`
`refine … ?_`	some hypotheses are proved afterwards	Recipe 2
`rw`, `simpa using`	the conclusion is an equation, or almost your goal	Recipe 6
`.1`, `.2`, `obtain`	the conclusion is an “and” or an “exists”	Section 5
combine	one theorem’s output is another’s input	Recipe 4

Directly, when your goal is the conclusion — `exact`:

LEAN 4

```
example (A B : Matrix (Fin 2) (Fin 3) ℚ) : (A + B).rank ≤ A.rank + B.rank :=
  matrix_rank_add_le A B
```

As a fact you then use — `have` and a tactic that finishes:

LEAN 4

```
example (A : Matrix (Fin 2) (Fin 3) ℚ) : (A + A).rank ≤ 2 * A.rank := by
  have := matrix_rank_add_le A A
  omega
```

Leaving some arguments for later — `refine` with `?_`, each `?_` becomes a new goal (Recipe 2 does this).

When the conclusion is an equation — `rw` rewrites with it; `simp only [...]` or `simpa using ...` let Lean adjust small differences of form (Recipe 6).

Specializing — give values to the variables (`wolstenholme_choose_modEq 7 …`) and take a part of an "and" with `.1` / `.2` (Section 5), or of an "exists" with `obtain ⟨v, hv⟩ := …`.

Combining two theorems — the output of one is the hypothesis of the other (Recipe 4).

Inside a longer proof that uses Mathlib — a theorem of the layer is an ordinary Lean theorem: mix it with Mathlib lemmas and any tactic.

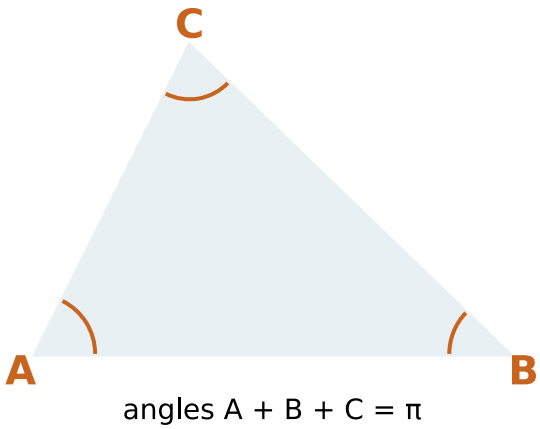
08 CHAPTER 8 🧪 Six recipes

Each recipe has the same parts: 🧩 the problem, 📜 the theorem, 💻 the code, ✅ what Lean checked, and 🎯 a variation to try. All the code is in the file `ReusableManualDemo.lean`, compiled against release `v0.15.0`; paste it after `open Formalizations.Reusable Matrix Polynomial` in a file that imports `Formalizations.Reusable`.

	AREA	THEOREM	YOU GET
1	📐 Geometry	`wolstenholme_defect_eq_sq_add_sq`	an inequality for every triangle
2	🧬 Polynomials	`card_roots_toFinset_ge_of_alternating_sign`	a lower bound on real roots
3	🧬 Polynomials	`newton_inequality_esymm_roots`	constraints on the coefficients
4	🔢 Linear algebra	`posSemidef_of_charpoly_alternating`, `psd_quadForm_eq_sum_rank_sq`	a sum-of-squares certificate
5	🔣 Number theory	`wolstenholme_choose_modEq`	a congruence modulo p^3
6	🔢 Linear algebra	`matrix_rank_add_rank_le_rank_fromBlocks`	a rank bound

RECIPE 1 an inequality of a triangle in two lines

Open card ↗



🧩 **The problem.** For the angles A, B, C of a triangle and any real numbers x, y, z: 2yz cos A + 2zx cos B + 2xy cos C ≤ x² + y² + z² (Wolstenholme's inequality).

📜 **The theorem.** `wolstenholme_defect_eq_sq_add_sq` (card R0002) says *why*: when A + B + C = π, the difference between the two sides is exactly a sum of two squares.

💻 **The code.**

LEAN 4

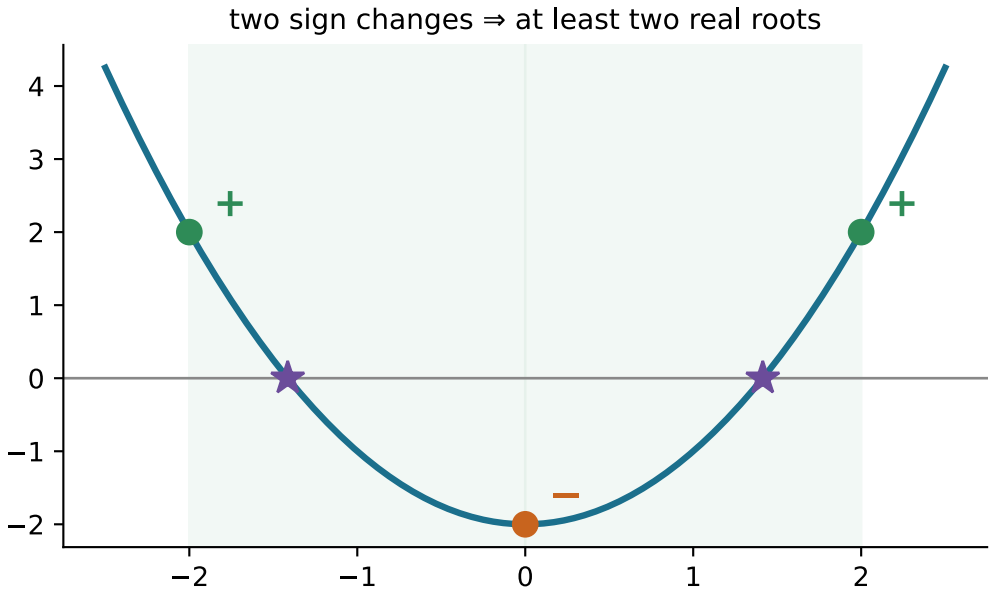
```
example (x y z A B C : ℝ) (h : A + B + C = Real.pi) :
    2 * y * z * Real.cos A + 2 * z * x * Real.cos B + 2 * x * y * Real.cos C
      ≤ x ^ 2 + y ^ 2 + z ^ 2 := by
  have key := wolstenholme_defect_eq_sq_add_sq x y z A B C h
  nlinarith [sq_nonneg (x - z * Real.cos B - y * Real.cos C),
    sq_nonneg (z * Real.sin B - y * Real.sin C)]
```

✅ **What Lean checked.**
`have key := …` puts the identity among the facts; `nlinarith` combines it with the fact that both squares are non-negative. The inequality holds for *every* triangle and *every* x, y, z.

🎯 **Try it.**
Set x = y = z = 1 to get cos A + cos B + cos C ≤ 3/2. Example 2 checks this special case in Lean.

RECIPE 2 real roots counted by sign changes

Open card ↗



🧩 **The problem.** Show that a polynomial has at least n distinct real roots without finding them.

📜 **The theorem.** `card_roots_toFinset_ge_of_alternating_sign` (card R0011): if a nonzero real polynomial changes sign between consecutive points of n + 1 increasing points, it has at least n distinct real roots, one in each gap.

💻 **The code.** For $X^2 - 2$ at the points −2, 0, 2:

LEAN 4

```
example : 2 ≤ ((X ^ 2 - C 2 : ℝ[X]).roots.toFinset.card) := by
  have hne : (X ^ 2 - C 2 : ℝ[X]) ≠ 0 := by
    intro h
    have := congrArg (Polynomial.eval 0) h
    simp at this
  refine card_roots_toFinset_ge_of_alternating_sign _ hne 2 ![-2, 0, 2] ?_ ?_
  · refine Fin.strictMono_iff_lt_succ.2 fun i ⇒ ?_
    fin_cases i <;> norm_num
  · intro i
    fin_cases i <;> norm_num
```

✅ **What Lean checked.**
The three things you supply: the polynomial is not zero (at 0 it is −2), the points −2 < 0 < 2 increase, and the values at consecutive points have opposite signs: 2·(−2) < 0 and (−2)·2 < 0.

🎯 **Try it.**
For $X^3 - 3X$ take the points −2, −1, 1, 2 (values −2, 2, −2, 2) and n = 3: three real roots.

RECIPE 3 the coefficients of a real-rooted polynomial

Open card ↗

🧩 **The problem.** If all the roots of a polynomial are real, its coefficients cannot be arbitrary.

📜 **The theorem.** `newton_inequality_esymm_roots` (card R0015), Newton's inequalities: for a monic real polynomial whose roots are all real (it *splits* over $\mathbb{R}$), the elementary symmetric functions e_k of the roots satisfy $e_k^2 \cdot C(n,k-1)\,C(n,k+1) \geq e_{k-1}\,e_{k+1} \cdot C(n,k)^2$.

💻 **The code.** The case k = 1, for every degree n ≥ 2:

LEAN 4

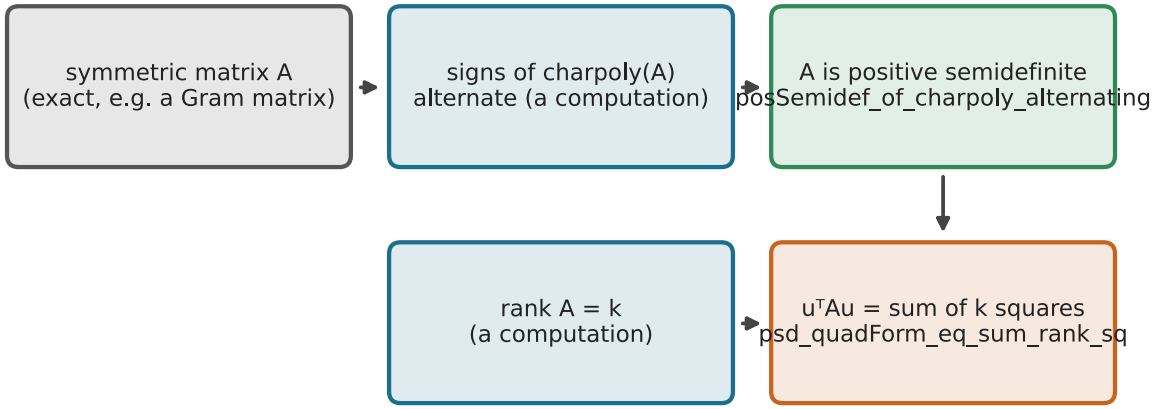
```
example (p : ℝ[X]) (hp : p.Splits) (hm : p.Monic) (h2 : 2 ≤ p.natDegree) :
    (p.roots.esymm 1) ^ 2 * ((p.natDegree.choose 0 : ℝ) * (p.natDegree.choose 2 : ℝ))
      ≥ (p.roots.esymm 0) * (p.roots.esymm 2) * ((p.natDegree.choose 1 : ℝ)) ^ 2 :=
  newton_inequality_esymm_roots hp hm le_rfl h2
```

✅ **What Lean checked.**
The theorem needs k ≥ 1 (here `le_rfl`, because k = 1) and k + 1 ≤ n (here `h2`). With $e_0 = 1$, e_1 = sum of the roots and e_2 = sum of the products of pairs, it reads $(n - 1)\, e_1^2 \geq 2n\, e_2$.

🎯 **Try it.**
For a quadratic $X^2 + bX + c$ with real roots, this is $b^2 \geq 4c$: the discriminant is non-negative.

RECIPE 4 a certificate that a quadratic form is a sum of k squares

Open card ↗



🧩 **The problem.** You have a symmetric matrix A, known exactly (for example a Gram matrix found by a computer search), and you want a *proof* that its quadratic form u^TAu is a sum of exactly k squares of linear forms.

📜 **The theorems.** Two steps:

1. `posSemidef_of_charpoly_alternating` (card R0032): if the coefficients of the characteristic polynomial of a symmetric real matrix alternate in sign, the matrix is positive semidefinite. Checking the signs of finitely many rational numbers is an exact computation.
2. `psd_quadForm_eq_sum_rank_sq` (card R0033): a positive semidefinite matrix of rank k has a quadratic form that is a sum of exactly k squares.

💻 The code.

LEAN 4

```
example {n : Type*} [Fintype n] [DecidableEq n] (A : Matrix n n ℝ) (hA : A.IsSymm)
    (halt : ∀ k ≤ A.charpoly.natDegree, 0 ≤ (-1 : ℝ) ^ (A.charpoly.natDegree - k) *
A.charpoly.coeff k)
    (k : ℕ) (hk : A.rank = k) :
    ∃ v : Fin k → (n → ℝ), ∀ u, u ⬝ᵥ (A *ᵥ u) = ∑ i, (v i ⬝ᵥ u) ^ 2 :=
  psd_quadForm_eq_sum_rank_sq (posSemidef_of_charpoly_alternating hA halt) hk
```

✅ What Lean checked.
The first theorem turns the sign condition `halt` into positive semidefiniteness; the second turns that and the rank into the k squares. This is the chain I use in my own research on sums of squares: a computer finds a candidate, an exact computation checks signs and rank, and these two theorems make the conclusion a theorem.

🎯 Try it.
For the matrix ((2, 1), (1, 2)) the characteristic polynomial is $X^2 - 4X + 3$: the signs +, −, + alternate and the rank is 2, so the form $2u_1^2 + 2u_1u_2 + 2u_2^2$ is a sum of two squares.

RECIPE 5 a congruence of Wolstenholme

Open card ↗

The example of Section 5 is this recipe: `wolstenholme_choose_modEq` (card R0026) with p = 7.

🎯 Try it.
With p = 11 and the second congruence `.2`: $C(22, 11) \equiv 2 \pmod{11^3}$.

RECIPE 6 rank bounds for block matrices

Open card ↗

🧩 **The problem.** Bound the rank of a matrix given in blocks, without computing it.

📜 **The theorem.** `matrix_rank_add_rank_le_rank_fromBlocks` (card R0023): over a field, the rank of a block upper-triangular matrix [[A, B], [0, D]] is at least rank A + rank D, whatever B is.

💻 The code.

LEAN 4

```
example (B : Matrix (Fin 2) (Fin 2) ℚ) :
    4 ≤ (fromBlocks (1 : Matrix (Fin 2) (Fin 2) ℚ) B 0 (1 : Matrix (Fin 2) (Fin 2)
ℚ)).rank := by
  have h := matrix_rank_add_rank_le_rank_fromBlocks (1 : Matrix (Fin 2) (Fin 2) ℚ) B
    (1 : Matrix (Fin 2) (Fin 2) ℚ)
  simpa only [rank_one, Fintype.card_fin] using h
```

✅ What Lean checked.

With identity blocks on the diagonal, rank A + rank D = 2 + 2 = 4: the 4 × 4 matrix is invertible, for every corner B.

🎯 Try it.

Replace the second identity block by a matrix of rank 1: the bound becomes 3.

09 Practical examples, start to finish

CHAPTER 9

Six problems as you would meet them, solved from the question to the checked proof. Each one shows a different way of working with the layer. All the code is in the file `ReusableManualPractice.lean`, compiled against release `v0.15.0`; it starts with `open Formalizations.Reusable Matrix Polynomial` and `open scoped Kronecker`.

	YOU WANT TO SHOW	YOU USE	THE IDEA
1	a cubic has three real roots	your own lemma on top of R0011	build once, reuse
2	cos A + cos B + cos C ≤ 3/2	R0002 with x = y = z = 1	specialise a general identity
3	the rank of a 6 × 6 tensor product	R0024	never write the matrix
4	B(7) ≡ 2 (mod 7)	R0031 with n = 0	rewrite, then compute
5	C(26, 13) ≡ 2 (mod 13^3)	R0026, second part	take the part you need
6	2 sin(π/3) · 2 sin(2π/3) = 3	R0006 with n = 3	let `exact_mod_cast` fix the casts

Example 1 — your own lemma, then a concrete cubic

The question. Does $X^3 - 3X + 1$ have three real roots? You will meet this kind of question again, so first write a small lemma of your own on top of the layer (card R0011): four increasing points with alternating signs give three distinct real roots. Then the cubic is one line.

LEAN 4

```
theorem three_real_roots (p : ℝ[X]) (hp : p ≠ 0) (a b c d : ℝ) (h1 : a < b) (h2 : b < c) (h3 :
c < d)
    (s1 : p.eval a * p.eval b < 0) (s2 : p.eval b * p.eval c < 0) (s3 : p.eval c * p.eval d <
0) :
    3 ≤ p.roots.toFinset.card := by
  refine card_roots_toFinset_ge_of_alternating_sign p hp 3 ![a, b, c, d] ?_ ?_
  · refine Fin.strictMono_iff_lt_succ.2 fun i ⇒ ?_
    fin_cases i
    · simpa using h1
    · simpa using h2
    · simpa using h3
  · intro i
    fin_cases i
    · simpa using s1
    · simpa using s2
    · simpa using s3

example : 3 ≤ ((X ^ 3 - C 3 * X + 1 : ℝ[X]).roots.toFinset.card) := by
  have hne : (X ^ 3 - C 3 * X + 1 : ℝ[X]) ≠ 0 := by
    intro h
    have := congrArg (Polynomial.eval 0) h
    simp at this
  exact three_real_roots _ hne (-2) 0 1 2 (by norm_num) (by norm_num) (by norm_num)
    (by norm_num) (by norm_num) (by norm_num)
```

What happened. `three_real_roots` turns the general theorem (points given as a vector `![a, b, c, d]`) into a statement with plain hypotheses `a < b`, `eval a p * eval b p < 0`, … that are easy to supply. For the cubic, `norm_num` evaluates the polynomial at −2, 0, 1, 2 (values −1, 1, −1, 3) and checks every hypothesis. *Lesson:* when you will use a theorem many times, wrap it once in the form your problems have.

Example 2 — a classical bound for every triangle

The question. Show cos A + cos B + cos C ≤ 3/2 for the angles of any triangle.

LEAN 4

```
example (A B C : ℝ) (h : A + B + C = Real.pi) : Real.cos A + Real.cos B + Real.cos C ≤ 3 / 2 :=
by
  have key := wolstenholme_defect_eq_sq_add_sq 1 1 1 A B C h
  nlinarith [key, sq_nonneg (1 - 1 * Real.cos B - 1 * Real.cos C), sq_nonneg (1 * Real.sin B -
1 * Real.sin C)]
```

What happened. The identity of card R0002 holds for all x, y, z; with x = y = z = 1 it says that 3 − 2(cos A + cos B + cos C) is a sum of two squares. `nlinarith` only has to add the two facts "a square is ≥ 0". *Lesson:* a general identity is a family of results; choosing the values is the proof.

Example 3 — the rank of a tensor product, without the matrix

The question. What is the rank of the Kronecker product of the 2 × 2 and 3 × 3 identity matrices?

LEAN 4

```
example : ((1 : Matrix (Fin 2) (Fin 2) ℚ) ⊗ₖ (1 : Matrix (Fin 3) (Fin 3) ℚ)).rank = 6 := by
  rw [matrix_rank_kronecker]
  simp [rank_one]
```

What happened. `rw [matrix_rank_kronecker]` (card R0024) replaces the rank of the 6 × 6 matrix by the product of two ranks, and `simp` knows that an identity of size n has rank n. *Lesson:* rewrite with the theorem first, then let `simp` finish the small computation.

Example 4 — Bell numbers modulo a prime

The question. The Bell number B(7) counts the partitions of a 7-element set (it is 877). What is it modulo 7?

LEAN 4

```
example : ((Nat.bell 7 : ℕ) : ZMod 7) = 2 := by
  have : Fact (Nat.Prime 7) := ⟨by norm_num⟩
  have h := touchard_bell_add_prime 7 0
  rw [zero_add, Nat.bell_zero, zero_add, Nat.bell_one] at h
  rw [h]
  rfl
```

What happened. Touchard's congruence (card R0031) needs to know that 7 is prime: the line `have : Fact …` provides it. With n = 0 it reads B(7) = B(0) + B(1) in `ZMod 7`; rewriting B(0) = B(1) = 1 leaves 1 + 1 = 2, which `rfl` checks. *Lesson:* some theorems take a hypothesis as an instance in square brackets; you give it with `have : Fact …`.

Example 5 — a central binomial coefficient modulo p^3

The question. Is C(26, 13) = 10 400 600 congruent to 2 modulo $13^3 = 2197$?

LEAN 4

```
example : Nat.choose 26 13 ≡ 2 [MOD 13 ^ 3] :=
  (wolstenholme_choose_modEq 13 (by norm_num) (by norm_num)).2
```

What happened. Wolstenholme's theorem (card R0026) gives two congruences joined by "and"; `.2` takes the second. Lean checks by itself that 2 · 13 = 26. *Lesson:* `.1` and `.2` pick the half of an "and" you need.

Example 6 — an exact trigonometric product

The question. Compute 2 sin(π/3) · 2 sin(2π/3) exactly.

LEAN 4

```
example : ∏ k ∈ Finset.Ico (1 : ℕ) 3, 2 * Real.sin (k * Real.pi / 3) = 3 := by
  have h := prod_two_sin_mul_pi_div 3 (by norm_num)
  exact_mod_cast h
```

What happened. Card R0006 says that the product of 2 sin(kπ/n) over k = 1, ..., n − 1 is n. The theorem is stated with a natural number n turned into a real one; `exact_mod_cast` removes those conversions so that the statement with n = 3 matches yours. *Lesson:* when the only difference is a cast between ℕ and ℝ, `exact_mod_cast` (or `push_cast`) bridges it.

From theorems to mathematical machines

A mathematical machine chains steps with explicit inputs and outputs. In Recipe 4, the goal is to prove that the quadratic form of A = ((2, 1), (1, 2)) is a sum of two squares.

Plan the calculation. From an Excovery checkout, ask the design planner for tools that take a rational Gram matrix and return rank information.

TERMINAL

```
python -m excovery disenar planificar --tengo gram_racional --quiero cota_rango
```

The planner proposes two checks by input and output type, one using the characteristic polynomial and one using the Schur complement. You run the tool. Registering individual Reusable cards as planner steps is future work.

Compute exactly. Run the characteristic-polynomial check on A.

TERMINAL

```
python -m excovery.charpoly_psd '[[2,1],[1,2]]'
```

The result gives symmetry, positive semidefiniteness, rank 2 and coefficients [1, −4, 3] in descending degree.

Check the chain in Lean. Prove symmetry and verify the alternating signs of $X^2 - 4X + 3$. These facts imply positive semidefiniteness (card R0032). The nonzero determinant 3 gives rank 2. With both facts, prove that $u^T Au$ is a sum of two squares of linear forms (card R0033).

Test a hypothesis. Excovery's `necesidad` tool searches for counterexamples to a weakened statement you model. Drop A + B + C = π from Recipe 1 and take A = B = C = 0 and x = y = z = 1. The claimed inequality becomes 6 ≤ 3. A search that finds no counterexample only tells you about the cases it tried.

10 CHAPTER 10 Every theorem of the layer, by area

The 82 theorems of release `v0.15.0`, in 9 areas, read from its manifest. 🔌 marks an `API` lemma, 🏛 a `showcase` theorem; each name links to its card, and each area gives the line that imports it.

> 🔄 **This list grows with every release.** The theorems available today — with their cards, search and examples — are always at https://karlesmarin.github.io/blueprints/reusable/. Follow the link from this PDF to see the current list.

🧩 Combinatorics 23

`import Formalizations.Reusable.Combinatorics`

R0008 🏛
pomerance_density_add_dvd_centralBinom
For every integer k ≥ 1, the set of n with n + k dividing the central binomial coefficient C(2n, n) has asymptotic density 1.

R0020 🏛
det_lapMatrix_submatrix_eq_card_spanningTrees
Kirchhoff's matrix-tree theorem: over an integral domain, deleting the row and column of any vertex from the Laplacian (Mathlib's lapMatrix) of a finite simple graph leaves a matrix whose determinant is the number of spanning trees.

R0028 🔌
card_biUnion_submodular
For an indexed family of finite sets t, the neighbourhood function s ↦ |s.biUnion t| is submodular: |N(s ∪ u)| + |N(s ∩ u)| ≤ |N(s)| + |N(u)|.

R0029 🔌
card_biUnion_union_inter_eq_of_tight
Under Hall's condition |v| ≤ |v.biUnion t| for all v, the union and the intersection of two tight sets (|s.biUnion t| = |s|) are tight.

R0030 🏛
bondy_exists_erase_injOn
Bondy's theorem: if X is a nonempty finite set and F a family of at most |X| distinct finite sets, some x ∈ X can be deleted from every member of F keeping them distinct.

R0035 🏛
godsil_gutman
Godsil–Gutman: the sum over all ±1 edge signings of the characteristic polynomials of the signed adjacency matrices is (number of signings) times the matching polynomial.

R0036 🏛
bass_determinant_commRing
Bass's determinant formula over any commutative ring: (1-u²)^|V| det(1-uB) = (1-u²)^|E| det(1-uA+u²(D-1)) with B the Hashimoto matrix.

R0040 🏛
bregman_permanent_le
Brégman's theorem (Minc's conjecture): for an n × n (0,1)-matrix A with row sums r_i, per A ≤ ∏ (r_i!)^(1/r_i); zero rows allowed (factor 1).

R0041 🔌
colorable_iff_forall_finite_induce
De Bruijn–Erdős: a graph G is n-colourable iff every finite induced subgraph is n-colourable (no finiteness hypothesis on the vertex type).

R0043 🏛
bruck_ryser_chowla
Bruck–Ryser–Chowla: if a finite projective plane of order n exists and n ≡ 1 or 2 (mod 4), then n is a sum of two squares (so no plane of order 6 or 14).

R0044 🔌
transversalMatroid_indep
For a finite family A : Fin t → Finset α, the independent finite sets of the transversal matroid presented by A are exactly the partial transversals of A (by Hall, the I with |S| ≤ |U_{x∈S} {i | x ∈ A i}| for all S ⊆ I).

R0046 🔌
orientedIncMatrix_mul_transpose
For a finite simple graph on a linearly ordered vertex type, over any ring, the oriented incidence matrix N satisfies N Nᵀ = D − A, the Laplacian (Mathlib's SimpleGraph.lapMatrix).

R0048
`ncard_choose_eq_le_two_mul_log`
Singmaster's bound: an integer a ≥ 2 occurs at most 2⌊log₂ a⌋ times in Pascal's triangle (the set of positions (n,k), k ≤ n, with C(n,k) = a is finite with at most that many elements; Singmaster states 2 + 2⌊log₂ a⌋).

R0068
`matchingPoly_forest_eq_charpoly`
The matching polynomial of a finite forest equals the characteristic polynomial of its real adjacency matrix.

R0069
`matchingPoly_splits`
Heilmann–Lieb real-rootedness: the matching polynomial of every finite simple graph splits over ℝ.

R0070
`matchingPoly_root_bound`
Heilmann–Lieb: if every vertex degree is at most Δ ≥ 2, then every real root of the matching polynomial lies in [−2√(Δ−1), 2√(Δ−1)].

R0071
`godsil_pathTree_identity`
Godsil's path-tree identity: μ(G) μ(T−r) = μ(G−u) μ(T), where T is the path tree at u, r is its root, and deletion takes the induced graph on the remaining vertices.

R0072
`matchingPoly_root_power_sum_eq_pathTree_walks`
Godsil's moment theorem: the sum of kth powers of the complex matching roots with multiplicity counts closed tree-like walks of length k, expressed by the root entries of the path-tree adjacency matrix powers.

R0078
`matchingPoly_derivative_eq_sum_vertex_deletions`
The derivative of the matching polynomial equals the sum of the matching polynomials of the induced graphs obtained by deleting one vertex at a time.

R0079
`treeLikeWalkCount_eq_trace_of_lt_girth`
Below the extended girth, the number of closed tree-like walks of length k equals the trace of the kth adjacency matrix power over ℕ.

R0080
`trace_hashimoto_pow`
The trace of the kth Hashimoto matrix power counts closed non-backtracking walks based at darts, cast into the coefficient ring.

R0081
`trace_sub_matchingPoly_root_power_sum_eq_trace_hashimoto`
For 1 ≤ k ≤ girth + 1, the adjacency trace minus the kth power sum of the complex matching roots, counted with multiplicity, equals the Hashimoto trace.

R0082
`ihara_Nk_explicit`
The series with coefficient tr(B^(k+1)) at X^k for the Hashimoto matrix B is given by differentiating Bass's determinant identity and inverting its unit factor.

Linear algebra 18

`import Formalizations.Reusable.LinearAlgebra`

R0009
`rank_fromBlocks_zero₂₁_le_card_add_rank`
Over a field, the rank of the block upper-triangular matrix [[A, B], [0, D]] is at most the number of rows of A plus the rank of D.

R0017
`matrix_rank_add_le`
Over a field, rank (A + B) ≤ rank A + rank B for matrices.

R0018
`det_mul_eq_sum_det_submatrix_mul_det_submatrix`
Cauchy–Binet: for A : m × n and B : n × m over a commutative ring (n linearly ordered), det (A B) is the sum over the |m|-subsets S of n of det A[:, S] · det B[S, :].

R0019
`derivative_det_eq_trace_adjugate_mul`
Jacobi's formula: for a square matrix A of polynomials over a commutative ring, derivative (det A) = trace (adjugate A · A.map derivative).

R0023
`matrix_rank_add_rank_le_rank_fromBlocks`
Over a field, rank A + rank D ≤ rank of the block upper-triangular matrix fromBlocks A B 0 D.

R0024
`matrix_rank_kronecker`
Over a field, the rank of the Kronecker product A ⊗ₖ B of two matrices is rank A · rank B.

R0025
`adjugate_apply_self_eq_det_principal_submatrix`
For a square matrix A over a commutative ring, indexed by any finite type, adjugate A i i is the determinant of the principal submatrix obtained by deleting row and column i.

R0032
`posSemidef_of_charpoly_alternating`
A real symmetric matrix whose characteristic polynomial has coefficients alternating in sign is positive semidefinite (the polynomial rule is a corollary of Descartes' rule of signs).

R0033
`psd_quadForm_eq_sum_rank_sq`
The quadratic form of a real positive semidefinite matrix of rank k is a sum of exactly k squares of linear forms.

R0034
`rank_fromBlocks_of_isUnit₁₁`
Over a field, a block matrix with invertible top-left block A has rank card m + rank (D - C A^{-1} B); with A positive definite and zero Schur complement it is PSD of rank card m.

R0037
`biquadratic_sos_transport`
For P_H(x, y) = (x_0^2 + x_1^2)(y·y) + 2x_0x_1(y·Hy) over a commutative ring and D^TD = 1, every sum-of-squares representation of P_H by bilinear forms x·(B_i y) transports to one of P_{D^THD} by the forms B_i D (same number of squares); signed-graph switching is D = diag(s), s_i^2 = 1.

R0038
`int_columns_of_transpose_mul_self`
If an integer m × n matrix A satisfies A^TA = I, every column j is ±e_{f(j)} for an injective f : n → m (signed unit vectors in distinct rows); in particular an integer vector with v·v = 1 is ±e_i. False over ℚ.

R0042
`schur_commuting_finrank_le`
Schur's bound: a subspace of M_n(K) over a field K whose elements pairwise commute has dimension ≤ ⌊n^2/4⌋ + 1 (sharp: R0056).

R0047
`hua_identity`
Hua's identity in a division ring: for a ≠ 0, b ≠ 0 and ab ≠ 1, a − (a^{-1} + (b^{-1} − a)$^{-1}$)$^{-1}$ = aba.

R0053
`derivative_coe_charpolyRev`
Newton's identity for traces: for a square matrix M over a commutative ring R, d/dX (charpolyRev M) = − charpolyRev M · Σ_k trace(M^(k+1)) X^k in R⟦X⟧, with charpolyRev M = det(1 − X • M).

R0056
`schur_commuting_finrank_attained`
Schur's bound is sharp: for every field K and n ≥ 1 there is a subspace of M_n(K) whose elements pairwise commute, of dimension exactly ⌊n^2/4⌋ + 1 (the companion of R0042).

R0058
`mul_add_mul_eq_trace_smul_add_trace_smul_fin_two`
For two-by-two matrices A and B over a commutative ring, AB + BA = tr(A) B + tr(B) A + (tr(AB) − tr(A)tr(B)) I.

R0061
`resolventSeries_diag_mul_coe_charpolyRev`
For a square matrix M over a commutative ring, the power series with coefficients $(M^k)_{ii}$ times the reversed characteristic polynomial of M equals the reversed characteristic polynomial of the principal submatrix deleting i.

Analysis 14

`import Formalizations.Reusable.Analysis`

R0004
`airy_eq_zeros_unbounded_below`
Every solution f of the Airy equation f'' = x f on ℝ has zeros unbounded below: for every M there is x < M with f x = 0.

R0005
`glasser_integral`
For a ≥ 0 and b > 0, ∫_{v>0} v^{-1/2} exp(−a/v − b v) dv = √(π/b) exp(−2√(ab)) (the modified Bessel K_{1/2} integral).

R0006
`prod_two_sin_mul_pi_div`
For every n ≥ 1, ∏_{k=1}^{n−1} 2 sin(kπ/n) = n.

R0016
`enestrom_kakeya_ne_zero`
The Eneström–Kakeya theorem: a polynomial ∑_{k ≤ n} a_k z^k with real coefficients a_0 ≥ a_1 ≥ ⋯ ≥ a_n > 0 has no zero in the open unit disc ‖z‖ < 1.

R0039
`hlawka_generalised`
Generalised Hlawka inequality: for n ≥ 3 vectors of a real inner product space, with d_k the sum of the norms of all k-element subsums, d_2 + ⋯ + d_{n-1} ≤ (2^(n-2) − 1)(d_1 + d_n); n = 3 is Hlawka's ‖x+y‖+‖y+z‖+‖z+x‖ ≤ ‖x‖+‖y‖+‖z‖+‖x+y+z‖.

R0051
`zielinski_zych_inequality`
The Zieliński–Zych inequality: for every α > 0 and x ≥ 0, 0 ≤ x^α − π^α/2 + (π^α/2) cos x + α π^(α−1) sin x; the analytic core of the fractional-moment speed limit π^α ≤ 2 M_α τ^α (α = 1 is Margolus–Levitin).

R0054
margolus_levitin
The Margolus–Levitin quantum speed limit ($\hbar = 1$): for a normalized population p over nonnegative energies E with mean Ebar, if the autocorrelation $\Sigma\, p_n\, e^{-i E_n \tau}$ vanishes at $\tau \geq 0$, then $\pi \leq 2 \cdot \text{Ebar} \cdot \tau$.

R0059
mandelstam_tamm_L1
For a probability distribution over real energy levels, if its autocorrelation vanishes at time $\tau \geq 0$, then $D_1\tau \geq 1$, where D_1 is the mean absolute energy deviation from the reference energy.

R0062
variance_mixture_kurtosis_identity
For independent standard normal Z and nonnegative V with finite second moment and positive mean, $\sqrt{V} \cdot Z$ has kurtosis 3 $E[V^2] / E[V]^2$ and excess kurtosis 3 $\mathrm{Var}(V) / E[V]^2 \geq 0$.

R0073
Li₂_add_Li₂_one_sub
Euler's reflection formula for $0 < x < 1$: $\mathrm{Li}_2(x) + \mathrm{Li}_2(1 - x) = \pi^2/6 - \log(x)\log(1 - x)$.

R0074
Li₂_landen
Landen's identity for $0 < x < 1/2$: $\mathrm{Li}_2(x) + \mathrm{Li}_2(x/(x - 1)) = -\log^2(1 - x)/2$.

R0075
Li₂_sq
Duplication for the real dilogarithm: $\mathrm{Li}_2(x^2) = 2(\mathrm{Li}_2(x) + \mathrm{Li}_2(-x))$ for $|x| \leq 1$.

R0076
rogersL_fiveterm
Abel's five-term relation for the Rogers dilogarithm on (0, 1): $L(x) + L(y) = L(xy) + L(x(1 - y)/(1 - xy)) + L(y(1 - x)/(1 - xy))$.

R0077
Li₂c_I
At the imaginary unit, the complex dilogarithm is $\mathrm{Li}_2(i) = -\pi^2/48 + i\,G$, where G is Catalan's constant.

Number theory 10

`import Formalizations.Reusable.NumberTheory`

R0026
wolstenholme_choose_modEq
Wolstenholme's theorem: for a prime $p \geq 5$, $C(2p - 1, p - 1) \equiv 1$ and $C(2p, p) \equiv 2 \pmod{p^3}$.

R0031
touchard_bell_add_prime
Touchard's congruence: for every prime p and every n, the Bell numbers satisfy $B(n + p) \equiv B(n) + B(n + 1) \pmod{p}$; in particular $B(p) \equiv 2 \pmod{p}$.

R0045
ded_reciprocity
Dedekind reciprocity: for coprime positive integers h, k, $s(h,k) + s(k,h) = (h^2 + k^2 + 1)/(12hk) - 1/4$, with $s(h,k) = \Sigma_{r<k} ((r/k))((hr/k))$ the Dedekind sum.

R0049
sum_card_divisors_mul_totient
The Dirichlet identity $\tau * \varphi = \sigma$: for every n, $\Sigma_{d|n} \tau(d)\, \varphi(n/d) = \sigma(n)$, with τ the number of divisors, φ Euler's totient and σ the sum of divisors.

R0057
reverse_cyclotomic
Cyclotomic reciprocity: for $n \geq 2$, the nth cyclotomic polynomial over $\mathbb{R}$ equals its coefficient reversal.

R0063
exists_fib_modEq_period
For every positive m, the Fibonacci sequence modulo m has a positive period valid from index zero.

R0064
even_of_fib_modEq_period
For every modulus $m > 2$, each positive period of the Fibonacci sequence modulo m is even.

R0065
fib_prime_eq_legendreSym
For every prime p, the pth Fibonacci number equals the Legendre symbol $(5/p)$ modulo p.

R0066
flynn_connolly
Infinitely many odd k divide the sum $F_1 + \cdots + F_k$. For odd primes $p \equiv 2$ modulo 3 and $p \equiv \pm 2$ modulo 5, both $k = F_{(2p)}$ and $k = F_{(4p)}$ are odd and satisfy $F_{(k+2)} \equiv 1$ modulo k.

R0067
amiot_interval
A cyclic interval of k consecutive residues modulo n, with $1 < k < n - 1$, cannot be preserved by multiplication by a unit other than ± 1.

Polynomials 9

`import Formalizations.Reusable.Polynomial`

R0007

splits_of_alternating_sign

A nonzero real polynomial of degree at most n that alternates sign at n + 1 strictly increasing points splits over ℝ (is real-rooted).

R0011

card_roots_toFinset_ge_of_alternating_sign

If a nonzero real polynomial takes strictly alternating signs at n + 1 strictly increasing points, it has at least n distinct real roots.

R0012

splits_of_isRoot_mem_Ioo_of_gaps

A nonzero real polynomial of degree at most n with a root in each of the n open gaps between n + 1 strictly increasing points splits over ℝ.

R0013

sturm_signVariations_sub_eq_card_roots

Sturm's theorem: for squarefree p ∈ ℝ[X] and a < b with p(a), p(b) ≠ 0, the sign variations of Mathlib's Polynomial.sturmSeq p p' at a minus those at b equal the number of distinct roots of p in (a, b].

R0014

budan_fourier_card_roots_le

The Budan–Fourier theorem: for nonzero p ∈ ℝ[X] and a < b with p(a), p(b) ≠ 0, the number of roots of p in (a, b] counted with multiplicity is at most V(a) − V(b), with even difference, where V(x) is the number of sign variations of p, p′, ..., p^(deg p) at x.

R0015

newton_inequality_esymm_roots

Newton's inequalities: for a monic real polynomial p of degree n that splits over ℝ and 1 ≤ k ≤ n − 1, e_k² · C(n,k−1) C(n,k+1) ≥ e_{k−1} e_{k+1} · C(n,k)², with e_j the elementary symmetric functions of the roots.

R0021

splits_derivative

If a real polynomial splits over ℝ (all its roots are real), so does its derivative.

R0022

splits_reverse

If a real polynomial splits over ℝ, so does its reverse (the reciprocal polynomial X^deg · p(1/X)).

R0050

chebyshev_S_docagne

d'Ocagne's identity for the Vieta–Fibonacci polynomials: over a commutative ring, for Mathlib's Polynomial.Chebyshev.S and all p, q : ℤ, S p · S (q−1) − S q · S (p−1) = − S (p−q−1); the case q = p+1 is S_sq_add_S_sq.

Group theory 4

`import Formalizations.Reusable.GroupTheory`

R0027

polya_redfield_card_mul_sum_orbit_weight

Pólya–Redfield enumeration theorem: for a finite group G acting on a finite set α and colour weights w, |G| · Σ over orbits of colourings of Π_a w(F a) equals Σ_g p_1^(fixed points of g) · Π over the cycles of g of p_length, where p_k = Σ_c w(c)^k.

R0052

separation_lemma_of_pretransitive

Separation lemma, transitive case: if G acts pretransitively on Ω and A, B are finite subsets with #A · #B < #Ω, some g ∈ G satisfies g • a ∉ B for all a ∈ A (no finiteness of G or Ω assumed).

R0055

praeger_bounded_movement

Praeger's theorem on bounded movement: if a group acts on Ω with no fixed point and every finite set moved off itself by some element has at most m ≥ 1 points, then Ω is finite, there are at most 2m − 1 orbits, every orbit has at most 3m points and |Ω| ≤ 3m + t − 1, t the number of orbits.

R0060

fanoGroup_card

The group of permutations of the seven points of the Fano plane that preserve its seven lines has order 168.

Geometry 2

`import Formalizations.Reusable.Geometry`

R0001

pappus_hexagon

Pappus's hexagon theorem in homogeneous coordinates over any commutative ring: if A, B, C are collinear and a, b, c are collinear, the meets Ab∩aB, Ac∩aC, Bc∩bC are collinear, with no non-degeneracy hypothesis.

R0002

wolstenholme_defect_eq_sq_add_sq

If $A + B + C = \pi$, then $x^2 + y^2 + z^2 - (2yz \cos A + 2zx \cos B + 2xy \cos C) = (x - z \cos B - y \cos C)^2 + (z \sin B - y \sin C)^2$ for all real x, y, z; hence Wolstenholme's inequality.

Algebraic geometry 1

`import Formalizations.Reusable.AlgebraicGeometry`

R0010

cotes_harmonic_mean_polar

Cotes' harmonic means theorem: if a line p + t v meets a real plane curve f = 0 in deg g real points (g(t) = f(p + t v) splits), the harmonic-mean point lies on the polar line $\nabla f(p)\cdot(q - p) = -d\, f(p)$ of p.

Measure theory 1

`import Formalizations.Reusable.MeasureTheory`

R0003

torricelli_trumpet

The solid of revolution of $y = 1/x$, $x \geq 1$ (Torricelli's trumpet, Gabriel's horn) has Lebesgue volume π, while its plane section $\{x \geq 1, |y| \leq 1/x\}$ has infinite area.

11 When something goes wrong

CHAPTER 11

`unknown identifier` — the import is missing or wrong. Check the *Import* line of the card, or use `import Formalizations.Reusable` for everything; write `open Formalizations.Reusable` (or the full name).

`type mismatch` — your statement is close to the theorem's but not equal: an `≥` written as `≤`, a coercion from `ℕ` to `ℝ` missing, a different order of factors. Compare with the signature on the card; `simpa`, `push_cast` and `ring_nf` bridge small differences.

`failed to synthesize` — a bracket argument (`[Fintype n]`) that Lean cannot find for your type; use a standard type (`Fin k`) or add the instance as a hypothesis.

Two Mathlibs, or a first build that never ends — the project was created with the `math` template, or `rev` points to another version. Recreate it as in Section 4.1.

`lake build` takes hours — use `lake lean <file>` (Section 4.1, step 4).

12 CHAPTER 12 📦 Versions, updates, citing and contact

- **Pin a release** (`rev = "v0.15.0"`). The notes of each release list its new theorems (notes of v0.15.0); names stay stable within a major series.
- **Updating**: change `rev` to the new tag and run `lake update`. Read the release notes first: a deprecated alias tells you the new name.
- **Prebuilt archives**: each release attaches compiled archives for Linux and Windows, which Lake downloads instead of compiling.
- **Trust**: continuous integration checks, on every push to the repository, that each theorem depends only on the standard axioms and that the usage examples compile; `#print axioms` lets you check it yourself.
- **Citing**: cite the repository and the release tag, and the paper or classical source given on the card.
- **Questions, errors, or a theorem you would like**: open an issue at https://github.com/karlesmarin/formalizations/issues.

I worked with an AI assistant on this guide and its examples; every Lean example in it is compiled against release `v0.15.0`.

13 CHAPTER 13 Quick reference

Install

```
lake +leanprover/lean4:v4.35.0 new myproject std.toml
# lakefile.toml
[[require]]
name = "formalizations"
git  = "https://github.com/karlesmarin/formalizations"
rev  = "v0.15.0"

lake update
lake lean Example.lean
```

Values of release v0.15.0; for a later one, its tag at releases and its `lean-toolchain`.

Find

- Showcase: https://karlesmarin.github.io/blueprints/reusable/
- `#check name` — the statement
- `#print axioms name` — the axioms
- `import Formalizations.Reusable` — every area

Use

`exact t …`	the goal is the conclusion
`have h := t …`	a fact for later tactics
`refine t … ?_`	hypotheses become goals
`rw [t]`, `simpa using t`	equations, small differences
`(t …).1`, `obtain`	parts of “and” / “exists”

Read a signature

- { } implicit — Lean deduces it
- [ ] instance — Lean finds it
- ( ) you supply it
- after the last : what you get

Errors

- `unknown identifier` → import / `open`
- `type mismatch` → compare with the card; `simpa`, `push_cast`
- `failed to synthesize` → a standard type such as `Fin k`
- build takes hours → `lake lean <file>`

Releases

- Pin `rev`: nothing changes until you change it
- Update: new tag in `rev`, then `lake update`
- Cite the repository, the tag and the source on the card
- Issues: https://github.com/karlesmarin/formalizations/issues