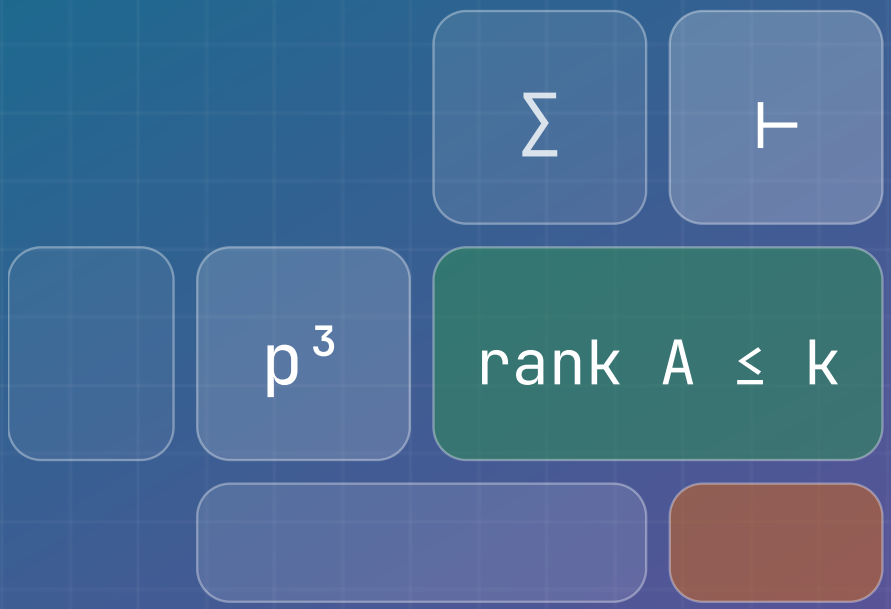
GUÍA PRÁCTICA · LEAN 4

La capa Reusable: guía práctica

Teoremas comprobados por máquina que puedes usar en tus propias pruebas de Lean 4: desde instalarlos hasta usar cada uno de ellos.

Ver el vídeo Simulaciones

v0.15.0 · 82 teoremas Lean 4 Mathlib comprobado por máquina código abierto



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

Índice

01	🎯	Qué podrás hacer	3
02	🧱	Qué es la capa Reusable	4
03	🧰	Antes de empezar	6
04	🚀	Instalar la capa	7
05	✍️	Tu primer teorema, palabra por palabra	10
06	🔎	Encontrar el teorema que necesitas	11
07	🛠️	Todas las formas de usar un teorema	13
08	🧪	Seis recetas	14
09	🧭	Ejemplos prácticos, de principio a fin	20
10	📚	Todos los teoremas de la capa, por área	24
11	🩺	Cuando algo falla	30
12	📦	Versiones, actualizaciones, cómo citar y contacto	31
13	🗂️	Referencia rápida	32

🧩 **Recetas** — problema, teorema, código, lo que Lean comprobó y algo para probar.

🔗 **Enlaces** — cada nombre de teorema enlaza con su ficha en la vitrina, siempre al día.

💻 **Código** — cada ejemplo de Lean está compilado con la versión.

🗂️ **Capítulo 13** — una referencia rápida de una página para tener a mano.

01 CAPÍTULO 1 🎯 Qué podrás hacer

Al terminar esta guía tendrás un proyecto de Lean que usa la capa Reusable, y sabrás encontrar, leer, aplicar y combinar cualquiera de sus teoremas. No hace falta leer sus pruebas: Lean ha comprobado cada una, y tú usas su enunciado como usarías un teorema de un libro.

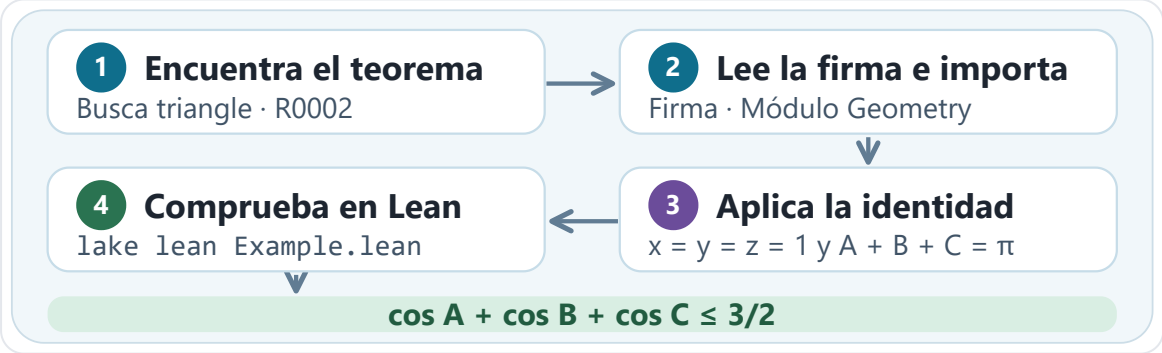
La capa sirve cuando Mathlib no tiene (todavía) el enunciado que necesitas, o lo tiene en una forma incómoda de aplicar. Tres cosas que hace por ti, todas resueltas en esta guía:

- 🔢 **Contar raíces reales** de un polinomio con unos pocos cambios de signo, sin calcular las raíces (Receta 2).
- ✅ **Certificar que una forma cuadrática es suma de k cuadrados**: dos teoremas convierten una comprobación de signos en la descomposición (Receta 4).
- 📐 **Demostrar una desigualdad clásica en dos líneas** cuando se tiene la identidad adecuada (Receta 1).

> 💡 **IDEA**
> **Todo el método en una frase.** Busca el teorema, dale sus argumentos, demuestra sus hipótesis y quédate con la parte de su conclusión que necesitas. El resto de la guía es esa frase en detalle.

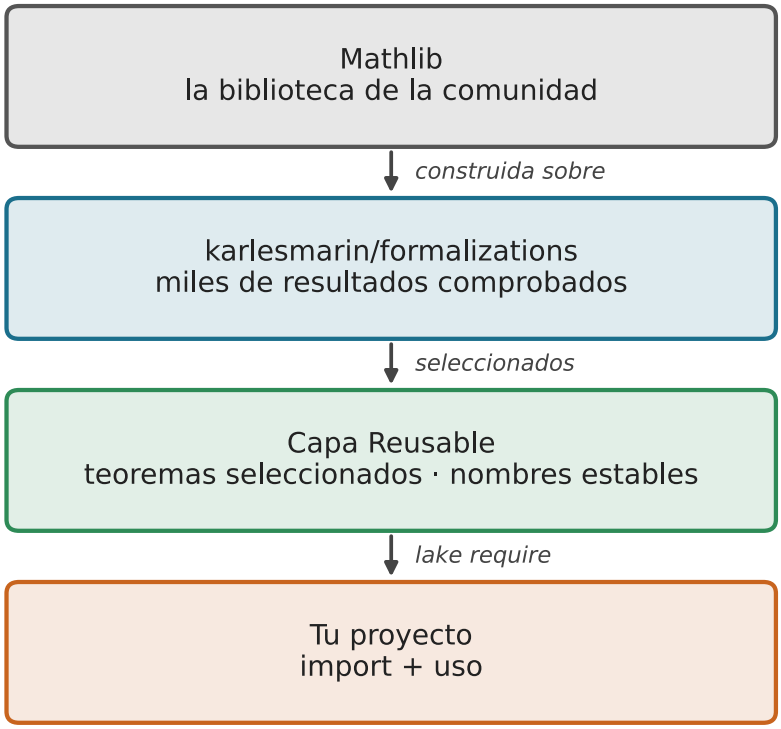
De la ficha a la prueba en Lean

¿Puedes demostrar que **cos A + cos B + cos C ≤ 3/2** para los ángulos de un triángulo? Este recorrido te lleva de la búsqueda a la prueba.



1. **Encuentra el teorema.** Busca `triangle` en la vitrina y abre la identidad de Wolstenholme (ficha R0002).
2. **Lee la firma y copia el import.** Empieza el fichero con `import Formalizations.Reusable.Geometry` y `open Formalizations.Reusable`. En la firma, `h` es la prueba que debes aportar de que A + B + C = π.
3. **Aplica la identidad.** La Receta 1 usa los dos cuadrados para demostrar la desigualdad general. En el Ejemplo 2, da los valores x = y = z = 1 y la prueba `h`. La identidad expresa 3 − 2(cos A + cos B + cos C) como suma de dos cuadrados.
4. **Comprueba la prueba.** Pega el Ejemplo 2 en `Example.lean`. La prueba usa `sq_nonneg` para cada cuadrado y `nlinarith` para deducir la cota. Ejecuta `lake lean Example.lean` en tu proyecto configurado. Si el comando termina correctamente, Lean ha aceptado la prueba.

02 CAPÍTULO 2 🧱 Qué es la capa Reusable



El repositorio `karlesmarin/formalizations` reúne miles de resultados comprobados por máquina, casi todos escritos para mis artículos de investigación. La **capa Reusable** es una parte pequeña y seleccionada: teoremas útiles fuera del artículo del que vienen, con nombres estables de la forma `Formalizations.Reusable.<nombre>`. La versión `v0.15.0`, la que describe esta edición, tiene 82 en 9 áreas.

Cada entrada de la capa:

- es un **teorema demostrado en el archivo**: la capa lo vuelve a exportar; no se copia ni se reescribe ninguna prueba;
- depende solo de los **axiomas estándar** de Lean (`propext`, `Classical.choice`, `Quot.sound`), comprobado en cada versión;
- tiene un **papel**: 🔌 `API`, un lema pensado para usarse dentro de otras pruebas, o 🏛️ `showcase`, un teorema clásico, completo en sí mismo, que se consulta más que se importa;
- tiene una **ficha** en la vitrina: el enunciado en palabras, la firma exacta en Lean, el import que necesita, un ejemplo compilado, de dónde viene y qué axiomas usa.

Lo que **no** es: no sustituye a Mathlib (se construye sobre Mathlib y fija una versión concreta de ella) y no es una colección de tácticas. Es un conjunto de teoremas.

📌 **NOTA**

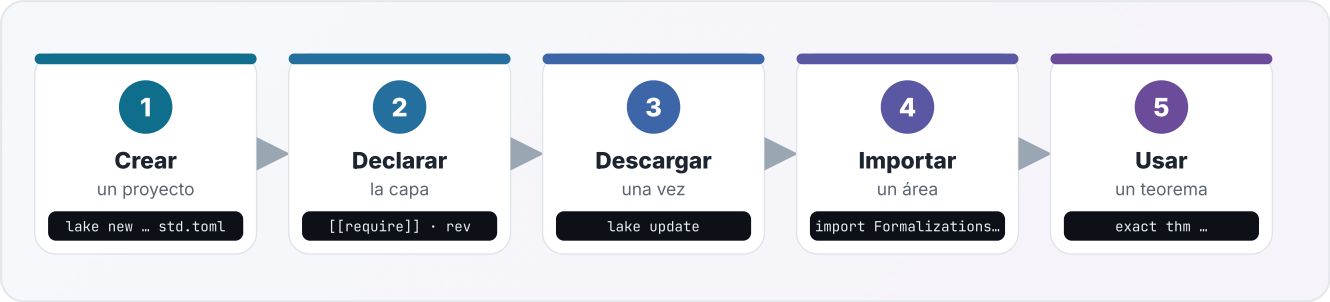
Estabilidad. Los nombres bajo `Formalizations.Reusable` no se retiran dentro de una serie principal. Si un enunciado tiene que cambiar de forma incompatible, el nombre antiguo se queda como alias obsoleto durante dos versiones.

03 CAPÍTULO 3 Antes de empezar

Necesitas:

- **Lean 4 con** `elan`, el gestor de versiones de Lean. Si nunca has instalado Lean, sigue https://lean-lang.org/install/; son unos minutos e instala también el soporte para VS Code.
- **git**.
- Unos **5 GB de disco**: Mathlib, ya compilada, se descarga una vez.
- **Lean básico**: qué significan `theorem`, `example`, `by`, `exact` y `have`. Si estas palabras son nuevas, los primeros capítulos de *Mathematics in Lean* (https://leanprover-community.github.io/mathematics_in_lean/) son el mejor comienzo; vuelve aquí después.

04 CAPÍTULO 4 Instalar la capa



4.1 En un proyecto nuevo, paso a paso

0 **Crear el proyecto.**

TERMINAL

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

> ⚠ **ATENCIÓN**
> Usa la plantilla **estándar** (`std.toml`), no `math` . La plantilla `math` añade su propia Mathlib, fijada en una versión distinta de la que usa esta capa, y acabarías con dos Mathlib que no concuerdan. Mathlib entra en tu proyecto a través de esta capa.

1 **Declarar la dependencia.** Abre `lakefile.toml` y añade al final:

LAKEFILE.TOML

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

`rev` fija la versión: tu proyecto seguirá funcionando igual hasta que la cambies.

> 📌 **NOTA**
>
> **De dónde salen los valores.** Esta edición de la guía se genera a partir de la versión `v0.15.0`: su etiqueta va en `rev`, y su versión de Lean (`leanprover/lean4:v4.35.0`, el fichero `lean-toolchain` en la etiqueta), en la línea de `lake new`. Para una versión posterior, toma su etiqueta de https://github.com/karlesmarin/formalizations/releases y el `lean-toolchain` de esa etiqueta; la vitrina da siempre los dos.

2 **Descargar.**

```
TERMINAL
lake update
```

Descarga la capa y Mathlib, y trae Mathlib ya compilada. La primera vez tarda (son varios gigabytes); las siguientes actualizaciones son rápidas.

3 **Un primer fichero.** Crea `Example.lean` en la carpeta del proyecto:

```
LEAN 4
import Formalizations.Reusable.NumberTheory

open Formalizations.Reusable

#check wolstenholme_choose_modEq
```

4 **Comprobarlo.**

```
TERMINAL
lake lean Example.lean
```

Lean escribe el enunciado del teorema: tu proyecto está listo. En Linux y Windows, Lake descarga la capa ya compilada; en macOS la compila una vez.

> ⚠️ **ATENCIÓN**
>
> Evita un `lake build` a secas en un proyecto de la plantilla estándar: su objetivo por defecto es un ejecutable, y construirlo compila a código nativo cada módulo de Mathlib importado, lo que lleva horas. Lo que quieres es `lake lean <fichero>`; si tus ficheros forman parte de una biblioteca de tu proyecto, `lake build <biblioteca>` hace lo mismo.

4.2 En un proyecto que ya tienes

Si tu proyecto usa **las mismas versiones de Lean y Mathlib** (las que dan `lean-toolchain` y `lake-manifest.json` del repositorio en la etiqueta que fijes; para `v0.15.0`, Lean `v4.35.0` y el commit `f0b4c81` de Mathlib), añade el bloque `[[require]]` del paso 1 a su `lakefile.toml` (o la línea `require` equivalente en un `lakefile.lean`) y ejecuta `lake update`. Si usa otra Mathlib, Lake avisará del conflicto: o pasas tu proyecto a estas versiones, o esperas a una versión de la capa para las tuyas.

4.3 Trabajar en VS Code

Abre la carpeta del proyecto en VS Code con la extensión de Lean 4. Al abrir `Example.lean`, el panel **infoview** muestra, en la posición del cursor, lo que queda por demostrar y el tipo de lo que hay bajo el cursor. Al pasar el ratón sobre un nombre como `wolstenholme_choose_modEq` aparece su enunciado; Ctrl-clic (Cmd-clic) lleva a donde está definido. Es la forma más rápida de leer un teorema mientras lo usas.

05 CAPÍTULO 5 ✍️ Tu primer teorema, palabra por palabra

Sustituye la línea `#check` de `Example.lean` por:

LEAN 4

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

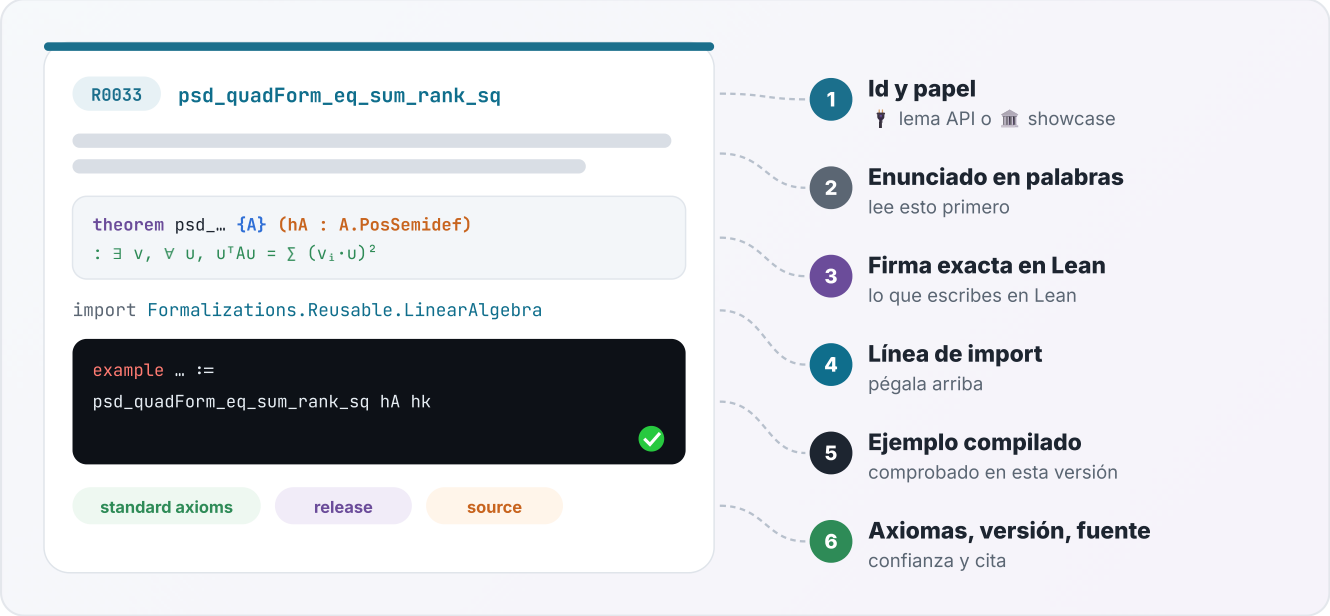
Vuelve a ejecutar `lake lean Example.lean`. Si no sale nada, Lean ha aceptado la prueba. Pieza a pieza:

- `example : …` enuncia algo sin darle nombre. Aquí: el coeficiente binomial C(13, 6) = 1716 es congruente con 1 módulo $7^3 = 343$.
- `wolstenholme_choose_modEq` es el teorema de la capa (Wolstenholme, 1862): *para un primo $p \geq 5$, $C(2p-1, p-1) \equiv 1 \pmod{p^3}$ y $C(2p, p) \equiv 2 \pmod{p^3}$.*
- `7` es el valor de p. Lean pide entonces las dos hipótesis del teorema: que 7 es primo y que $5 \leq 7$. Cada `(by norm_num)` demuestra una de ellas calculando.
- El teorema da dos congruencias unidas por «y»; `.1` toma la primera, que es justo nuestro enunciado ($2 \cdot 7 - 1 = 13$ y $7 - 1 = 6$).

06 CAPÍTULO 6 🔎 Encontrar el teorema que necesitas

La vitrina (https://karlesmarin.github.io/blueprints/reusable/) lista todos los teoremas con un buscador. Busca con palabras («rank», «real roots», «triangle», «congruence») o recórrela por áreas. Cada ficha muestra:

- el enunciado en palabras y la **firma exacta en Lean**;
- el **import**: el módulo que escribes después de `import` (el área del teorema, por ejemplo `Formalizations.Reusable.LinearAlgebra`); `import Formalizations.Reusable` importa todas las áreas a la vez;
- un **ejemplo compilado**: código que Lean comprobó en esta versión, listo para copiar;
- los **axiomas** que usa, la **versión** (con sus versiones de Lean y de Mathlib) y el **artículo** o la fuente clásica de la que viene.



El manifiesto `Formalizations/Reusable/manifest.toml` tiene la misma información como datos, una entrada por teorema, por si prefieres buscar con tus propios programas. La **sección 10** de esta guía lista todos los teoremas por área.

Desde Lean puedes preguntar directamente:

LEAN 4

```
#check psd_quadForm_eq_sum_rank_sq        -- su enunciado
#print axioms psd_quadForm_eq_sum_rank_sq -- en qué axiomas se apoya
```

Leer una firma

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
```

{ } implícito: Lean lo deduce [ ] instancia: Lean la encuentra ( ) lo das tú ▇ la conclusión: lo que obtienes

- Los argumentos **entre llaves** `{n}`, `{A}`, `{k}` son *implícitos*: Lean los deduce de los demás; no los escribes.
- Los argumentos **entre corchetes** `[Fintype n]` son *instancias*: Lean las encuentra solo para los tipos habituales (`Fin 3`, `ℕ`, `ℝ`, ...).
- Los argumentos **entre paréntesis** `(hA : …)`, `(hk : …)` son los que das tú: aquí, una prueba de que `A` es semidefinida positiva y una prueba de que su rango es `k`.
- Lo que sigue a los últimos dos puntos es la **conclusión**: lo que obtienes.

07 CAPÍTULO 7 🛠️ Todas las formas de usar un teorema

Un mismo teorema puede entrar en una prueba de varias maneras. Aquí van con un solo teorema, `matrix_rank_add_le` (el rango de una suma es como mucho la suma de los rangos).

TÁCTICA	ÚSALA CUANDO	EJEMPLO
`exact`	el objetivo es justo la conclusión	`exact matrix_rank_add_le A B`
`have`	quieres la conclusión como dato para después	`have h := matrix_rank_add_le A A`
`refine … ?_`	algunas hipótesis se demuestran después	Receta 2
`rw`, `simpa using`	la conclusión es una ecuación, o casi tu objetivo	Receta 6
`.1`, `.2`, `obtain`	la conclusión es un «y» o un «existe»	Sección 5
combinar	la salida de un teorema es la entrada de otro	Receta 4

Directamente, cuando lo que quieres demostrar es su conclusión — `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
```

Como un dato que después usas — `have` y una táctica que termina:

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
```

Dejando argumentos para después — `refine` con `?_`: cada `?_` se convierte en un objetivo nuevo (la Receta 2 lo hace).

Cuando la conclusión es una ecuación — `rw` reescribe con ella; `simp only [...]` o `simpa using ...` dejan que Lean ajuste pequeñas diferencias de forma (Receta 6).

Particularizando — da valores a las variables (`wolstenholme_choose_modEq 7 …`) y toma una parte de un «y» con `.1` / `.2` (sección 5), o de un «existe» con `obtain ⟨v, hv⟩ := …`.

Combinando dos teoremas — la salida de uno es la hipótesis del otro (Receta 4).

Dentro de una prueba más larga que usa Mathlib — un teorema de la capa es un teorema de Lean como cualquier otro: combínalo con lemas de Mathlib y con cualquier táctica.

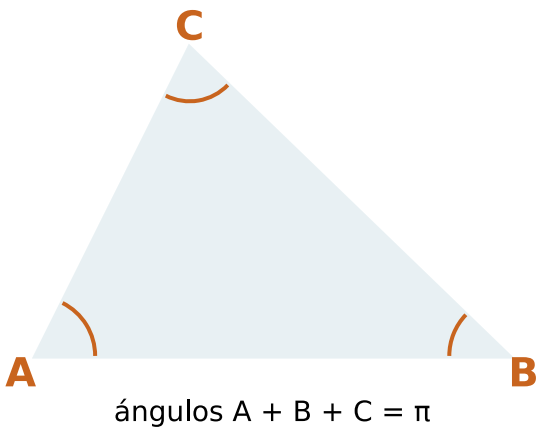
08 CAPÍTULO 8 🧪 Seis recetas

Cada receta tiene las mismas partes: 🧩 el problema, 📜 el teorema, 💻 el código, ✅ lo que Lean comprobó y 🎯 una variación para probar. Todo el código está en el fichero `ReusableManualDemo.lean`, compilado con la versión `v0.15.0`; pégalo después de `open Formalizations.Reusable Matrix Polynomial` en un fichero que importe `Formalizations.Reusable`.

	ÁREA	TEOREMA	OBTIENES
1	📐 Geometría	`wolstenholme_defect_eq_sq_add_sq`	una desigualdad para todo triángulo
2	🧬 Polinomios	`card_roots_toFinset_ge_of_alternating_sign`	una cota inferior de raíces reales
3	🧬 Polinomios	`newton_inequality_esymm_roots`	restricciones sobre los coeficientes
4	🔢 Álgebra lineal	`posSemidef_of_charpoly_alternating`, `psd_quadForm_eq_sum_rank_sq`	un certificado de suma de cuadrados
5	🔣 Teoría de números	`wolstenholme_choose_modEq`	una congruencia módulo p^3
6	🔢 Álgebra lineal	`matrix_rank_add_rank_le_rank_fromBlocks`	una cota de rango

RECETA 1 una desigualdad del triángulo en dos líneas

Abrir ficha ↗



🧩 **El problema.** Para los ángulos A, B, C de un triángulo y cualesquiera reales x, y, z: $2yz\cos A + 2zx\cos B + 2xy\cos C \le x^2 + y^2 + z^2$ (desigualdad de Wolstenholme).

📜 **El teorema.** `wolstenholme_defect_eq_sq_add_sq` (ficha R0002) dice *por qué*: cuando A + B + C = π, la diferencia entre los dos lados es exactamente una suma de dos cuadrados.

💻 **El código.**

LEAN 4

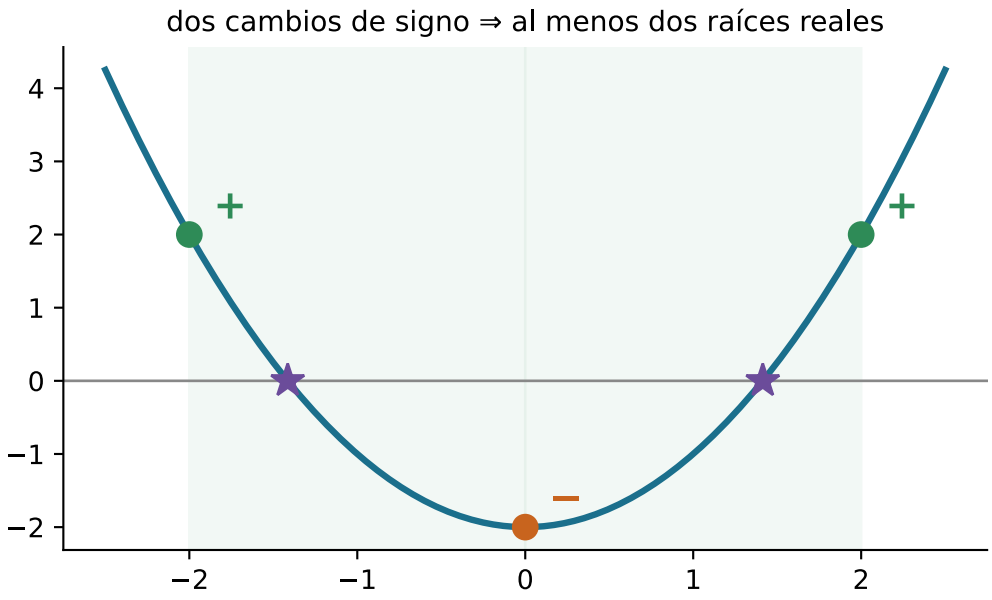
```
example (x y z A B C : ℝ) (h : A + B + C = Real.pi) :
    2 * y * z * Real.cos A + 2 * z * x * Real.cos B + 2 * x * y * Real.cos C
      ≤ x ^ 2 + y ^ 2 + z ^ 2 := by
  have key := wolstenholme_defect_eq_sq_add_sq x y z A B C h
  nlinarith [sq_nonneg (x - z * Real.cos B - y * Real.cos C),
    sq_nonneg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

✅ **Lo que Lean comprobó.**
`have key := …` pone la identidad entre los datos; `nlinarith` la combina con que los dos cuadrados son no negativos. La desigualdad vale para *todo* triángulo y *todos* x, y, z.

🎯 **Pruébalo.**
Toma x = y = z = 1 para obtener cos A + cos B + cos C ≤ 3/2. El Ejemplo 2 comprueba este caso particular en Lean.

RECETA 2 raíces reales contadas por cambios de signo

Abrir ficha ↗



🧩 **El problema.** Demostrar que un polinomio tiene al menos n raíces reales distintas sin encontrarlas.

📜 **El teorema.** `card_roots_toFinset_ge_of_alternating_sign` (ficha R0011): si un polinomio real no nulo cambia de signo entre puntos consecutivos de n + 1 puntos crecientes, tiene al

menos n raíces reales distintas, una en cada hueco.

💻 **El código.** Para $X^2 - 2$ en los puntos −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
```

✅ **Lo que Lean comprobó.**
Las tres cosas que das tú: el polinomio no es cero (en 0 vale −2), los puntos −2 < 0 < 2 crecen, y los valores en puntos consecutivos tienen signos opuestos: 2·(−2) < 0 y (−2)·2 < 0.

🎯 **Pruébalo.**
Para $X^3 - 3X$ toma los puntos −2, −1, 1, 2 (valores −2, 2, −2, 2) y n = 3: tres raíces reales.

RECETA 3 los coeficientes de un polinomio con raíces reales

Abrir ficha ↗

🧩 **El problema.** Si todas las raíces de un polinomio son reales, sus coeficientes no pueden ser cualesquiera.

📜 **El teorema.** `newton_inequality_esymm_roots` (ficha R0015), las desigualdades de Newton: para un polinomio real mónico con todas sus raíces reales (se *descompone* sobre $\mathbb{R}$), las funciones simétricas elementales e_k de las raíces cumplen $e_k^2 \cdot C(n,k-1)\,C(n,k+1) \geq e_{k-1}\,e_{k+1} \cdot C(n,k)^2$.

💻 **El código.** El caso k = 1, para todo grado 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
```

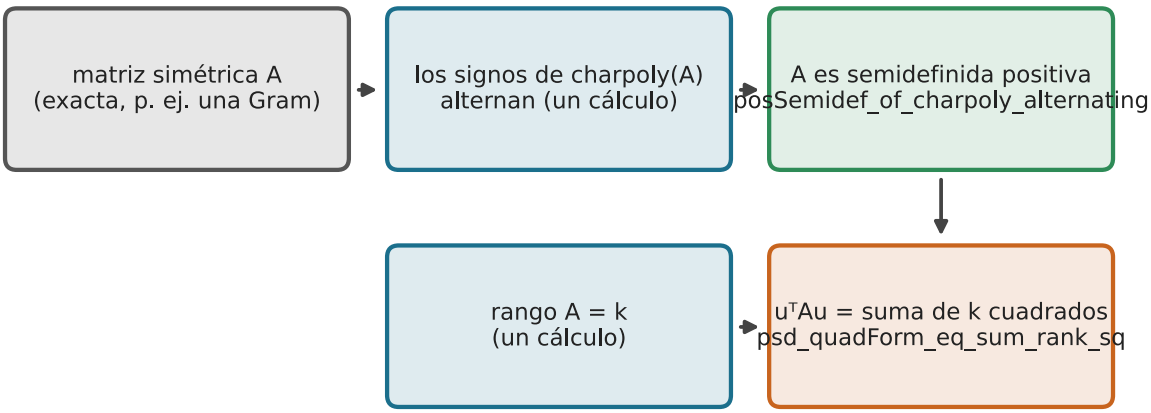
✅ **Lo que Lean comprobó.**
El teorema pide k ≥ 1 (aquí `le_rfl`, porque k = 1) y k + 1 ≤ n (aquí `h2`). Con $e_0 = 1$, e_1 = suma de las raíces y e_2 = suma de los productos de dos en dos, se lee $(n - 1)\, e_1^2 \geq 2n\, e_2$.

🎯 **Pruébalo.**
Para un polinomio de grado dos $X^2 + bX + c$ con raíces reales, es $b^2 \geq 4c$: el discriminante no es negativo.

RECETA 4

un certificado de que una forma cuadrática es suma de k cuadrados

Abrir ficha ↗



🧩 **El problema.** Tienes una matriz simétrica A, conocida de forma exacta (por ejemplo una matriz de Gram encontrada por un programa), y quieres una *prueba* de que su forma cuadrática $u^T A u$ es suma de exactamente k cuadrados de formas lineales.

📜 **Los teoremas.** Dos pasos:

1. `posSemidef_of_charpoly_alternating` (ficha R0032): si los coeficientes del polinomio característico de una matriz real simétrica alternan de signo, la matriz es semidefinida positiva. Comprobar los signos de un número finito de racionales es un cálculo exacto.
2. `psd_quadForm_eq_sum_rank_sq` (ficha R0033): una matriz semidefinida positiva de rango k tiene una forma cuadrática que es suma de exactamente k cuadrados.

💻 El código.

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
```

✅ Lo que Lean comprobó.
El primer teorema convierte la condición de signos `halt` en que la matriz es semidefinida positiva; el segundo convierte eso y el rango en los k cuadrados. Es la cadena que uso en mi propia investigación sobre sumas de cuadrados: un programa encuentra un candidato, un cálculo exacto comprueba signos y rango, y estos dos teoremas convierten la conclusión en un teorema.

🎯 Pruébalo.
Para la matriz ((2, 1), (1, 2)) el polinomio característico es $X^2 - 4X + 3$: los signos +, −, + alternan y el rango es 2, así que la forma $2u_1^2 + 2u_1u_2 + 2u_2^2$ es suma de dos cuadrados.

RECETA 5 una congruencia de Wolstenholme

Abrir ficha ↗

El ejemplo de la sección 5 es esta receta: `wolstenholme_choose_modEq` (ficha R0026) con p = 7.

🎯 Pruébalo.
Con p = 11 y la segunda congruencia `.2`: $C(22, 11) \equiv 2 \pmod{11^3}$.

RECETA 6 cotas de rango para matrices por bloques

Abrir ficha ↗

🧩 **El problema.** Acotar el rango de una matriz dada por bloques, sin calcularlo.

📜 **El teorema.** `matrix_rank_add_rank_le_rank_fromBlocks` (ficha R0023): sobre un cuerpo, el rango de una matriz triangular superior por bloques [[A, B], [0, D]] es al menos rango A + rango D, sea cual sea B.

💻 El código.

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
```

✅ Lo que Lean comprobó.

Con bloques identidad en la diagonal, rango A + rango D = 2 + 2 = 4: la matriz 4 × 4 es invertible, sea cual sea la esquina B.

🎯 Pruébalo.

Cambia el segundo bloque identidad por una matriz de rango 1: la cota pasa a ser 3.

09 CAPÍTULO 9 Ejemplos prácticos, de principio a fin

Seis problemas tal como te los encontrarías, resueltos desde la pregunta hasta la prueba comprobada. Cada uno muestra una forma distinta de trabajar con la capa. Todo el código está en el fichero `ReusableManualPractice.lean`, compilado con la versión `v0.15.0`; empieza con `open Formalizations.Reusable Matrix Polynomial` y `open scoped Kronecker`.

	QUIERES DEMOSTRAR	USAS	LA IDEA
1	una cúbica tiene tres raíces reales	tu propio lema sobre R0011	construir una vez, reutilizar
2	cos A + cos B + cos C ≤ 3/2	R0002 con x = y = z = 1	particularizar una identidad general
3	el rango de un producto tensorial 6 × 6	R0024	no escribir nunca la matriz
4	B(7) ≡ 2 (mod 7)	R0031 con n = 0	reescribir y calcular
5	C(26, 13) ≡ 2 (mod 13^3)	R0026, segunda parte	quedarte con la parte que necesitas
6	2 sen(π/3) · 2 sen(2π/3) = 3	R0006 con n = 3	dejar que `exact_mod_cast` arregle las conversiones

Ejemplo 1 — tu propio lema, y después una cúbica concreta

La pregunta. ¿Tiene $X^3 - 3X + 1$ tres raíces reales? Te encontrarás más preguntas así, de modo que primero escribe un lema tuyo sobre la capa (ficha R0011): cuatro puntos crecientes con signos alternos dan tres raíces reales distintas. Después la cúbica es una línea.

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)
```

Qué ha pasado. `three_real_roots` convierte el teorema general (con los puntos dados como un vector `![a, b, c, d]`) en un enunciado con hipótesis sencillas `a < b`, `eval a p * eval b p < 0`, … fáciles de dar. Para la cúbica, `norm_num` evalúa el polinomio en −2, 0, 1, 2 (valores −1, 1, −1, 3) y comprueba cada hipótesis. *Lección:* si vas a usar un teorema muchas veces, envuélvelo una vez en la forma que tienen tus problemas.

Ejemplo 2 — una cota clásica para todo triángulo

La pregunta. Demostrar cos A + cos B + cos C ≤ 3/2 para los ángulos de cualquier triángulo.

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)]
```

Qué ha pasado. La identidad de la ficha R0002 vale para todos x, y, z; con x = y = z = 1 dice que 3 − 2(cos A + cos B + cos C) es suma de dos cuadrados. `nlinarith` solo tiene que añadir los dos datos «un cuadrado es ≥ 0». *Lección:* una identidad general es una familia de resultados; elegir los valores es la prueba.

Ejemplo 3 — el rango de un producto tensorial, sin la matriz

La pregunta. ¿Cuál es el rango del producto de Kronecker de las identidades 2 × 2 y 3 × 3?

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]
```

Qué ha pasado. `rw [matrix_rank_kronecker]` (ficha R0024) cambia el rango de la matriz 6 × 6 por el producto de dos rangos, y `simp` sabe que una identidad de tamaño n tiene rango n. *Lección:* reescribe primero con el teorema y deja que `simp` termine el cálculo pequeño.

Ejemplo 4 — números de Bell módulo un primo

La pregunta. El número de Bell B(7) cuenta las particiones de un conjunto de 7 elementos (vale 877). ¿Cuánto vale módulo 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
```

Qué ha pasado. La congruencia de Touchard (ficha R0031) necesita saber que 7 es primo: lo da la línea `have : Fact …`. Con n = 0 dice B(7) = B(0) + B(1) en `ZMod 7`; al reescribir B(0) = B(1) = 1 queda 1 + 1 = 2, que comprueba `rfl`. *Lección:* algunos teoremas piden una hipótesis como instancia entre corchetes; se da con `have : Fact …`.

Ejemplo 5 — un coeficiente binomial central módulo p^3

La pregunta. ¿Es C(26, 13) = 10 400 600 congruente con 2 módulo $13^3 = 2197$?

LEAN 4

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

Qué ha pasado. El teorema de Wolstenholme (ficha R0026) da dos congruencias unidas por «y»; `.2` toma la segunda. Lean comprueba solo que 2 · 13 = 26. *Lección:* `.1` y `.2` eligen la mitad de un «y» que necesitas.

Ejemplo 6 — un producto trigonométrico exacto

La pregunta. Calcular de forma exacta 2 sen(π/3) · 2 sen(2π/3).

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
```

Qué ha pasado. La ficha R0006 dice que el producto de 2 sen(kπ/n) para k = 1, ..., n − 1 es n. El teorema está enunciado con un natural n convertido en real; `exact_mod_cast` quita esas conversiones para que el enunciado con n = 3 coincida con el tuyo. *Lección:* cuando la única diferencia es una conversión entre ℕ y ℝ, `exact_mod_cast` (o `push_cast`) la salva.

De teoremas a máquinas matemáticas

Una máquina matemática encadena pasos con entradas y salidas explícitas. En la Receta 4, el objetivo es demostrar que la forma cuadrática de A = ((2, 1), (1, 2)) es suma de dos cuadrados.

Planifica el cálculo. Desde el repositorio de Excovery, pide a la mesa de diseño herramientas que reciban una matriz de Gram racional y den información sobre su rango.

TERMINAL

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

La mesa propone dos comprobaciones según el tipo de entrada y salida, por polinomio característico o por complemento de Schur. Tú ejecutas la herramienta. Registrar cada ficha Reusable es una ampliación futura.

Calcula con aritmética exacta. Ejecuta la comprobación por polinomio característico sobre A.

TERMINAL

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

El resultado da simetría, semidefinitud positiva, rango 2 y los coeficientes [1, −4, 3] en grado decreciente.

Comprueba la cadena en Lean. Demuestra la simetría y la alternancia de signos de $X^2 - 4X + 3$. De ahí sale la semidefinitud positiva (ficha R0032). El determinante no nulo, 3, da rango 2. Con ambos datos, demuestra que $u^T Au$ es una suma de dos cuadrados de formas lineales (ficha R0033).

Pon a prueba una hipótesis. La herramienta `necesidad` de Excovery busca contraejemplos a un enunciado debilitado que tú modelas. Quita A + B + C = π de la Receta 1 y toma A = B = C = 0 y x = y = z = 1. La desigualdad se convierte en 6 ≤ 3. Una búsqueda sin contraejemplos solo informa sobre los casos que examinó.

CAPÍTULO 10

10 Todos los teoremas de la capa, por área

Los 82 teoremas de la versión `v0.15.0`, en 9 áreas, leídos de su manifiesto. 🔌 marca un lema `API`, 🏛️ un teorema `showcase`; cada nombre enlaza con su ficha, y cada área da la línea que la importa.

> 🔄 **Esta lista crece con cada versión.** Los teoremas disponibles hoy — con sus fichas, el buscador y los ejemplos — están siempre en https://karlesmarin.github.io/blueprints/reusable/. Sigue el enlace desde este PDF para ver la lista al día.

🧩 Combinatoria 23

`import Formalizations.Reusable.Combinatorics`

R0008 🏛️
`pomerance_density_add_dvd_centralBinom`
Para todo entero k ≥ 1, el conjunto de los n tales que n + k divide al coeficiente binomial central C(2n, n) tiene densidad asintótica 1.

R0020 🏛️
`det_lapMatrix_submatrix_eq_card_spanningTrees`
Teorema de Kirchhoff de la matriz y los árboles: sobre un dominio de integridad, al quitar la fila y la columna de cualquier vértice de la laplaciana (lapMatrix de Mathlib) de un grafo simple finito queda una matriz cuyo determinante es el número de árboles generadores.

R0028 🔌
`card_biUnion_submodular`
Para una familia indexada de conjuntos finitos t, la función de vecindad s ↦ |s.biUnion t| es submodular: |N(s ∪ u)| + |N(s ∩ u)| ≤ |N(s)| + |N(u)|.

R0029 🔌
`card_biUnion_union_inter_eq_of_tight`
Bajo la condición de Hall |v| ≤ |v.biUnion t| para todo v, la unión y la intersección de dos conjuntos ajustados (|s.biUnion t| = |s|) son ajustadas.

R0030 🏛️
`bondy_exists_erase_injOn`
Teorema de Bondy: si X es un conjunto finito no vacío y F una familia de como mucho |X| conjuntos finitos distintos, se puede quitar algún x ∈ X de todos los miembros de F sin que dejen de ser distintos.

R0035 🏛️
`godsil_gutman`
Godsil–Gutman: la suma, sobre todas las asignaciones de signos ±1 a las aristas, de los polinomios característicos de las matrices de adyacencia con signo es (el número de asignaciones) por el polinomio de emparejamientos.

R0036 🏛️
`bass_determinant_commRing`
Fórmula del determinante de Bass sobre cualquier anillo conmutativo: (1−u²)^|V| det(1−uB) = (1−u²)^|E| det(1−uA+u²(D−1)), con B la matriz de Hashimoto.

R0040 🏛️
`bregman_permanent_le`
Teorema de Brégman (conjetura de Minc): para una matriz A de ceros y unos de n × n con sumas por filas r_i, per A ≤ ∏ (r_i!)^(1/r_i); se admiten filas nulas (factor 1).

R0041 🔌
`colorable_iff_forall_finite_induce`
De Bruijn–Erdős: un grafo G es n-coloreable si y solo si todo subgrafo inducido finito es n-coloreable (sin hipótesis de finitud sobre el tipo de vértices).

R0043 🏛️
`bruck_ryser_chowla`
Bruck–Ryser–Chowla: si existe un plano proyectivo finito de orden n y n ≡ 1 o 2 (mod 4), entonces n es suma de dos cuadrados (así que no hay plano de orden 6 ni 14).

R0044 🔌
`transversalMatroid_indep`
Para una familia finita A : Fin t → Finset α, los conjuntos finitos independientes del matroide transversal presentado por A son exactamente los transversales parciales de A (por Hall, los I con |S| ≤ |⋃_{x∈S} {i | x ∈ A i}| para todo S ⊆ I).

R0046 🔌
`orientedIncMatrix_mul_transpose`
Para un grafo simple finito sobre un tipo de vértices linealmente ordenado, sobre cualquier anillo, la matriz de incidencia orientada N cumple N Nᵀ = D − A, la laplaciana (SimpleGraph.lapMatrix de Mathlib).

R0048
`ncard_choose_eq_le_two_mul_log`
Cota de Singmaster: un entero a ≥ 2 aparece como mucho 2⌊log₂ a⌋ veces en el triángulo de Pascal (el conjunto de posiciones (n,k), k ≤ n, con C(n,k) = a es finito con como mucho esos elementos; Singmaster da 2 + 2⌊log₂ a⌋).

R0068
`matchingPoly_forest_eq_charpoly`
El polinomio de emparejamientos de un bosque finito coincide con el polinomio característico de su matriz de adyacencia real.

R0069
`matchingPoly_splits`
Teorema de Heilmann–Lieb: el polinomio de emparejamientos de todo grafo simple finito se descompone en factores lineales sobre ℝ.

R0070
`matchingPoly_root_bound`
Teorema de Heilmann–Lieb: si el grado de cada vértice es como mucho Δ ≥ 2, toda raíz real del polinomio de emparejamientos está en [−2√(Δ−1), 2√(Δ−1)].

R0071
`godsil_pathTree_identity`
Identidad del árbol de caminos de Godsil: μ(G) μ(T−r) = μ(G−u) μ(T), donde T es el árbol de caminos desde u, r es su raíz y las eliminaciones son subgrafos inducidos.

R0072
`matchingPoly_root_power_sum_eq_pathTree_walks`
Teorema de momentos de Godsil: la suma, con multiplicidad, de las potencias k-ésimas de las raíces complejas del polinomio de emparejamientos cuenta los paseos cerrados de tipo árbol de longitud k mediante las entradas raíz–raíz de las potencias de la matriz de adyacencia del árbol de caminos.

R0078
`matchingPoly_derivative_eq_sum_vertex_deletions`
La derivada del polinomio de emparejamientos es la suma de los polinomios de emparejamientos de los subgrafos inducidos obtenidos al eliminar un vértice cada vez.

R0079
`treeLikeWalkCount_eq_trace_of_lt_girth`
Por debajo de la cintura extendida, el número de paseos cerrados de tipo árbol de longitud k coincide con la traza de la potencia k-ésima de la matriz de adyacencia sobre ℕ.

R0080
`trace_hashimoto_pow`
La traza de la potencia k-ésima de la matriz de Hashimoto cuenta los paseos cerrados sin retroceso basados en aristas dirigidas, convertidos al anillo de coeficientes.

R0081
`trace_sub_matchingPoly_root_power_sum_eq_trace_hashimoto`
Para 1 ≤ k ≤ cintura + 1, la traza de adyacencia menos la suma de potencias k-ésimas de las raíces complejas del polinomio de emparejamientos, con multiplicidad, coincide con la traza de Hashimoto.

R0082
`ihara_Nk_explicit`
La serie cuyo coeficiente de X^k es tr(B^(k+1)), para la matriz de Hashimoto B, se obtiene derivando la identidad del determinante de Bass e invirtiendo su factor unidad.

Álgebra lineal 18

`import Formalizations.Reusable.LinearAlgebra`

R0009
`rank_fromBlocks_zero₂₁_le_card_add_rank`
Sobre un cuerpo, el rango de la matriz triangular superior por bloques [[A, B], [0, D]] es como mucho el número de filas de A más el rango de D.

R0017
`matrix_rank_add_le`
Sobre un cuerpo, rango (A + B) ≤ rango A + rango B para matrices.

R0018
`det_mul_eq_sum_det_submatrix_mul_det_submatrix`
Cauchy–Binet: para A : m × n y B : n × m sobre un anillo conmutativo (n linealmente ordenado), det (A B) es la suma sobre los subconjuntos S de n con |m| elementos de det A[:, S] · det B[S, :].

R0019
`derivative_det_eq_trace_adjugate_mul`
Fórmula de Jacobi: para una matriz cuadrada A de polinomios sobre un anillo conmutativo, derivada (det A) = traza (adjunta A · A.map derivada).

R0023
`matrix_rank_add_rank_le_rank_fromBlocks`
Sobre un cuerpo, rango A + rango D ≤ rango de la matriz triangular superior por bloques fromBlocks A B 0 D.

R0024
`matrix_rank_kronecker`
Sobre un cuerpo, el rango del producto de Kronecker A ⊗ₖ B de dos matrices es rango A · rango B.

R0025
`adjugate_apply_self_eq_det_principal_submatrix`
Para una matriz cuadrada A sobre un anillo conmutativo, indexada por cualquier tipo finito, adjunta A i i es el determinante de la submatriz principal que resulta de quitar la fila y la columna i.

R0032
`posSemidef_of_charpoly_alternating`
Una matriz real simétrica cuyo polinomio característico tiene coeficientes de signos alternos es semidefinida positiva (la regla para polinomios es un corolario de la regla de los signos de Descartes).

R0033
`psd_quadForm_eq_sum_rank_sq`
La forma cuadrática de una matriz real semidefinida positiva de rango k es suma de exactamente k cuadrados de formas lineales.

R0034
`rank_fromBlocks_of_isUnit₁₁`
Sobre un cuerpo, una matriz por bloques con bloque superior izquierdo A invertible tiene rango card m + rango (D − C A^{-1} B); con A definida positiva y complemento de Schur nulo es semidefinida positiva de rango card m.

R0037
`biquadratic_sos_transport`
Para P_H(x, y) = (x_0^2 + x_1^2)(y·y) + 2x_0x_1(y·Hy) sobre un anillo conmutativo y D^TD = 1, toda representación de P_H como suma de cuadrados de formas bilineales x·(B_i y) se transporta a una de P_{D^THD} con las formas B_i D (el mismo número de cuadrados); el cambio de signos de un grafo con signo es D = diag(s), s_i^2 = 1.

R0038
`int_columns_of_transpose_mul_self`
Si una matriz entera A de m × n cumple A^TA = I, cada columna j es ±e_{f(j)} para una f : n → m inyectiva (vectores unitarios con signo en filas distintas); en particular un vector entero con v·v = 1 es ±e_i. Falso sobre ℚ.

R0042
`schur_commuting_finrank_le`
Cota de Schur: un subespacio de M_n(K) sobre un cuerpo K cuyos elementos conmutan dos a dos tiene dimensión ≤ ⌊n^2/4⌋ + 1 (óptima: R0056).

R0047
`hua_identity`
Identidad de Hua en un anillo de división: para a ≠ 0, b ≠ 0 y ab ≠ 1, a − (a^{-1} + (b^{-1} − $a)^{-1})^{-1}$ = aba.

R0053
`derivative_coe_charpolyRev`
Identidad de Newton para trazas: para una matriz cuadrada M sobre un anillo conmutativo R, d/dX (charpolyRev M) = − charpolyRev M · Σ_k traza(M^(k+1)) X^k en R⟦X⟧, con charpolyRev M = det(1 − X • M).

R0056
`schur_commuting_finrank_attained`
La cota de Schur es óptima: para todo cuerpo K y n ≥ 1 hay un subespacio de M_n(K) cuyos elementos conmutan dos a dos, de dimensión exactamente ⌊n^2/4⌋ + 1 (el compañero de R0042).

R0058
`mul_add_mul_eq_trace_smul_add_trace_smul_fin_two`
Para matrices A y B de tamaño 2 × 2 sobre un anillo conmutativo, AB + BA = tr(A) B + tr(B) A + (tr(AB) − tr(A)tr(B)) I.

R0061
`resolventSeries_diag_mul_coe_charpolyRev`
Para una matriz cuadrada M sobre un anillo conmutativo, la serie de potencias cuyos coeficientes son $(M^k)_{ii}$, multiplicada por el polinomio característico invertido de M, coincide con el polinomio característico invertido de la submatriz principal obtenida al eliminar i.

Análisis 14

`import Formalizations.Reusable.Analysis`

R0004
`airy_eq_zeros_unbounded_below`
Toda solución f de la ecuación de Airy f'' = x f en ℝ tiene ceros no acotados inferiormente: para todo M hay un x < M con f x = 0.

R0005
`glasser_integral`
Para a ≥ 0 y b > 0, ∫_{v>0} v^{-1/2} exp(−a/v − b v) dv = √(π/b) exp(−2√(ab)) (la integral de Bessel modificada K_{1/2}).

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

R0016
enestrom_kakeya_ne_zero
Teorema de Eneström–Kakeya: un polinomio ∑_{k ≤ n} a_k z^k de coeficientes reales a_0 ≥ a_1 ≥ ⋯ ≥ a_n > 0 no tiene ceros en el disco unidad abierto ‖z‖ < 1.

R0039
hlawka_generalised
Desigualdad de Hlawka generalizada: para n ≥ 3 vectores de un espacio real con producto interno, con d_k la suma de las normas de todas las subsumas de k elementos, d_2 + ⋯ + d_{n−1} ≤ (2^(n−2) − 1)(d_1 + d_n); el caso n = 3 es la de Hlawka ‖x+y‖+‖y+z‖+‖z+x‖ ≤ ‖x‖+‖y‖+‖z‖+‖x+y+z‖.

R0051
zielinski_zych_inequality
La desigualdad de Zieliński–Zych: para todo α > 0 y x ≥ 0, 0 ≤ x^α − π^α/2 + (π^α/2) cos x + α π^(α−1) sin x; el núcleo analítico del límite de velocidad de momentos fraccionarios π^α ≤ 2 M_α τ^α (α = 1 es Margolus–Levitin).

R0054
margolus_levitin
El límite de velocidad cuántico de Margolus–Levitin ($\hbar = 1$): para una población normalizada p sobre energías no negativas E de media Ebar, si la autocorrelación Σ p_n e^(−i E_n τ) se anula en τ ≥ 0, entonces π ≤ 2 · Ebar · τ.

R0059
mandelstam_tamm_L1
Para una distribución de probabilidad sobre niveles de energía reales, si su autocorrelación se anula en un tiempo τ ≥ 0, entonces D_1τ ≥ 1, donde D_1 es la desviación absoluta media respecto de la energía de referencia.

R0062
variance_mixture_kurtosis_identity
Para Z normal estándar y V no negativa independientes, con segundo momento finito y media positiva, √V · Z tiene curtosis 3 E[V^2] / E[V]² y exceso de curtosis 3 Var(V) / E[V]² ≥ 0.

R0073
Li₂_add_Li₂_one_sub
Fórmula de reflexión de Euler para 0 < x < 1: Li_2(x) + Li_2(1 − x) = π²/6 − log(x) log(1 − x).

R0074
Li₂_landen
Identidad de Landen para 0 < x < 1/2: Li_2(x) + Li_2(x/(x − 1)) = −log²(1 − x)/2.

R0075
Li₂_sq
Fórmula de duplicación del dilogaritmo real: Li_2(x^2) = 2(Li_2(x) + Li_2(−x)) si |x| ≤ 1.

R0076
rogersL_fiveterm
Relación de cinco términos de Abel para el dilogaritmo de Rogers en (0, 1): L(x) + L(y) = L(xy) + L(x(1 − y)/(1 − xy)) + L(y(1 − x)/(1 − xy)).

R0077
Li₂c_I
En la unidad imaginaria, el dilogaritmo complejo vale Li_2(i) = −π²/48 + i G, donde G es la constante de Catalan.

Teoría de números 10

`import Formalizations.Reusable.NumberTheory`

R0026
wolstenholme_choose_modEq
Teorema de Wolstenholme: para un primo p ≥ 5, C(2p − 1, p − 1) ≡ 1 y C(2p, p) ≡ 2 (mod p^3).

R0031
touchard_bell_add_prime
Congruencia de Touchard: para todo primo p y todo n, los números de Bell cumplen B(n + p) ≡ B(n) + B(n + 1) (mod p); en particular B(p) ≡ 2 (mod p).

R0045
ded_reciprocity
Reciprocidad de Dedekind: para enteros positivos coprimos h, k, s(h,k) + s(k,h) = (h^2 + k^2 + 1)/(12hk) − 1/4, con s(h,k) = ∑_{r<k} ((r/k))((hr/k)) la suma de Dedekind.

R0049
sum_card_divisors_mul_totient
La identidad de Dirichlet τ ∗ φ = σ: para todo n, ∑_{d|n} τ(d) φ(n/d) = σ(n), con τ el número de divisores, φ la función de Euler y σ la suma de divisores.

R0057
reverse_cyclotomic
Reciprocidad ciclotómica: para n ≥ 2, el n-ésimo polinomio ciclotómico sobre ℝ coincide con el polinomio obtenido al invertir el orden de sus coeficientes.

R0063
exists_fib_modEq_period
Para todo m positivo, la sucesión de Fibonacci módulo m tiene un período positivo válido desde el índice cero.

R0064
even_of_fib_modEq_period
Para todo módulo m > 2, cada período positivo de la sucesión de Fibonacci módulo m es par.

R0065
fib_prime_eq_legendreSym
Para todo primo p, el p-ésimo número de Fibonacci coincide módulo p con el símbolo de Legendre (5/p).

R0066
flynn_connolly
Hay infinitos k impares que dividen la suma F_1 + ⋯ + F_k. Si p es primo impar, p ≡ 2 (mod 3) y p ≡ ±2 (mod 5), tanto k = F_(2p) como k = F_(4p) son impares y satisfacen F_(k+2) ≡ 1 (mod k).

R0067
amiot_interval
Un intervalo cíclico de k residuos consecutivos módulo n, con 1 < k < n − 1, no puede conservarse al multiplicarlo por una unidad distinta de ±1.

Polinomios 9

`import Formalizations.Reusable.Polynomial`

R0007
splits_of_alternating_sign
Un polinomio real no nulo de grado como mucho n que alterna de signo en n + 1 puntos estrictamente crecientes se descompone sobre ℝ (tiene todas sus raíces reales).

R0011
card_roots_toFinset_ge_of_alternating_sign
Si un polinomio real no nulo toma signos estrictamente alternos en n + 1 puntos estrictamente crecientes, tiene al menos n raíces reales distintas.

R0012
splits_of_isRoot_mem_Ioo_of_gaps
Un polinomio real no nulo de grado como mucho n con una raíz en cada uno de los n huecos abiertos entre n + 1 puntos estrictamente crecientes se descompone sobre ℝ.

R0013
sturm_signVariations_sub_eq_card_roots
Teorema de Sturm: para p ∈ ℝ[X] libre de cuadrados y a < b con p(a), p(b) ≠ 0, las variaciones de signo de Polynomial.sturmSeq p p' de Mathlib en a menos las de b son el número de raíces distintas de p en (a, b].

R0014
budan_fourier_card_roots_le
Teorema de Budan–Fourier: para p ∈ ℝ[X] no nulo y a < b con p(a), p(b) ≠ 0, el número de raíces de p en (a, b] contadas con multiplicidad es como mucho V(a) − V(b), con diferencia par, donde V(x) es el número de variaciones de signo de p, p', ..., p^(deg p) en x.

R0015
newton_inequality_esymm_roots
Desigualdades de Newton: para un polinomio real mónico p de grado n que se descompone sobre ℝ y 1 ≤ k ≤ n − 1, e_k² · C(n,k−1) C(n,k+1) ≥ e_{k−1} e_{k+1} · C(n,k)², con e_j las funciones simétricas elementales de las raíces.

R0021
splits_derivative
Si un polinomio real se descompone sobre ℝ (todas sus raíces son reales), también su derivada.

R0022
splits_reverse
Si un polinomio real se descompone sobre ℝ, también su recíproco (el polinomio X^grado · p(1/X)).

R0050
chebyshev_S_docagne
Identidad de d'Ocagne para los polinomios de Vieta–Fibonacci: sobre un anillo conmutativo, para Polynomial.Chebyshev.S de Mathlib y todos p, q : ℤ, S p · S (q−1) − S q · S (p−1) = − S (p−q−1); el caso q = p+1 es S_sq_add_S_sq.

Teoría de grupos 4

`import Formalizations.Reusable.GroupTheory`

R0027
polya_redfield_card_mul_sum_orbit_weight
Teorema de enumeración de Pólya–Redfield: para un grupo finito G que actúa sobre un conjunto finito α y pesos de colores w, |G| · Σ sobre las órbitas de coloraciones de Π_a w(F a) es igual a Σ_g p_1^(puntos fijos de g) · Π sobre los ciclos de g de p_longitud, con p_k = Σ_c w(c)^k.

R0052
separation_lemma_of_pretransitive
Lema de separación, caso transitivo: si G actúa de forma pretransitiva sobre Ω y A, B son subconjuntos finitos con #A · #B < #Ω, algún g ∈ G cumple g • a ∉ B para todo a ∈ A (sin suponer finitos G ni Ω).

R0055
praeger_bounded_movement
Teorema de Praeger del movimiento acotado: si un grupo actúa sobre Ω sin puntos fijos y todo conjunto finito que algún elemento saca de sí mismo tiene como mucho m ≥ 1 puntos, entonces Ω es finito, hay como mucho 2m − 1 órbitas, cada órbita tiene como mucho 3m puntos y |Ω| ≤ 3m + t − 1, con t el número de órbitas.

R0060
fanoGroup_card
El grupo de permutaciones de los siete puntos del plano de Fano que conserva sus siete rectas tiene orden 168.

Geometría 2

`import Formalizations.Reusable.Geometry`

R0001

pappus_hexagon

Teorema del hexágono de Pappus en coordenadas homogéneas sobre cualquier anillo conmutativo: si A, B, C están alineados y a, b, c están alineados, los cortes Ab∩aB, Ac∩aC, Bc∩bC están alineados, sin hipótesis de no degeneración.

R0002

wolstenholme_defect_eq_sq_add_sq

Si A + B + C = π, entonces $x^2 + y^2 + z^2 - (2yz \cos A + 2zx \cos B + 2xy \cos C) = (x - z \cos B - y \cos C)^2 + (z \sin B - y \sin C)^2$ para todos x, y, z reales; de ahí la desigualdad de Wolstenholme.

Geometría algebraica 1

`import Formalizations.Reusable.AlgebraicGeometry`

R0010

cotes_harmonic_mean_polar

Teorema de las medias armónicas de Cotes: si una recta p + t v corta una curva plana real f = 0 en deg g puntos reales (g(t) = f(p + t v) se descompone), el punto de media armónica está en la polar $\nabla f(p)\cdot(q - p) = -d\, f(p)$ de p.

Teoría de la medida 1

`import Formalizations.Reusable.MeasureTheory`

R0003

torricelli_trumpet

El sólido de revolución de y = 1/x, x ≥ 1 (la trompeta de Torricelli, el cuerno de Gabriel) tiene volumen de Lebesgue π, mientras que su sección plana {x ≥ 1, |y| ≤ 1/x} tiene área infinita.

11 CAPÍTULO 11 Cuando algo falla

`unknown identifier` — falta el import o es otro. Mira la línea *Import* de la ficha, o usa `import Formalizations.Reusable` para todo; escribe `open Formalizations.Reusable` (o el nombre completo).

`type mismatch` — tu enunciado se parece al del teorema pero no es igual: un `≥` escrito como `≤`, falta pasar de `ℕ` a `ℝ`, otro orden de factores. Compáralo con la firma de la ficha; `simpa`, `push_cast` y `ring_nf` salvan diferencias pequeñas.

`failed to synthesize` — un argumento entre corchetes (`[Fintype n]`) que Lean no encuentra para tu tipo; usa un tipo estándar (`Fin k`) o añade la instancia como hipótesis.

Dos Mathlib, o una primera compilación que no acaba — el proyecto se creó con la plantilla `math`, o `rev` apunta a otra versión. Vuelve a crearlo como en la sección 4.1.

`lake build` tarda horas — usa `lake lean <fichero>` (sección 4.1, paso 4).

CAPÍTULO 12

12 📦 Versiones, actualizaciones, cómo citar y contacto

- **Fija una versión** (`rev = "v0.15.0"`). Las notas de cada versión listan sus teoremas nuevos (notas de v0.15.0); los nombres se mantienen dentro de una serie principal.
- **Actualizar**: cambia `rev` a la etiqueta nueva y ejecuta `lake update`. Lee antes las notas de la versión: un alias obsoleto te dice el nombre nuevo.
- **Archivos precompilados**: cada versión adjunta archivos compilados para Linux y Windows, que Lake descarga en lugar de compilar.
- **Confianza**: la integración continua comprueba, en cada push al repositorio, que cada teorema depende solo de los axiomas estándar y que los ejemplos de uso compilan; `#print axioms` te deja comprobarlo a ti.
- **Cómo citar**: cita el repositorio y la etiqueta de la versión, y el artículo o la fuente clásica que da la ficha.
- **Preguntas, errores o un teorema que te gustaría tener**: abre una incidencia en https://github.com/karlesmarin/formalizations/issues.

Trabajé con un asistente de IA en esta guía y sus ejemplos; cada ejemplo de Lean que aparece en ella está compilado con la versión `v0.15.0`.

13 CAPÍTULO 13 Referencia rápida

🚀 Instalar

```
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
```

Valores de la versión v0.15.0; para una posterior, su etiqueta en releases y su `lean-toolchain`.

🔎 Buscar

- Vitrina: https://karlesmarin.github.io/blueprints/reusable/
- `#check nombre` — el enunciado
- `#print axioms nombre` — los axiomas
- `import Formalizations.Reusable` — todas las áreas

✍️ Usar

`exact t …`	el objetivo es la conclusión
`have h := t …`	un dato para tácticas posteriores
`refine t … ?_`	las hipótesis pasan a objetivos
`rw [t]`, `simpa using t`	ecuaciones, diferencias pequeñas
`(t …).1`, `obtain`	partes de «y» / «existe»

📄 Leer una firma

- { } implícito — Lean lo deduce
- [ ] instancia — Lean la encuentra
- ( ) lo das tú
- tras los últimos : lo que obtienes

🩺 Errores

- `unknown identifier` → import / `open`
- `type mismatch` → compara con la ficha; `simpa`, `push_cast`
- `failed to synthesize` → un tipo estándar como `Fin k`
- compilar tarda horas → `lake lean <fichero>`

📦 Versiones

- Fija `rev`: nada cambia hasta que lo cambies
- Actualizar: etiqueta nueva en `rev` y `lake update`
- Cita el repositorio, la etiqueta y la fuente de la ficha
- Incidencias: https://github.com/karlesmarin/formalizations/issues