

Guía de diseño de preguntas con qbank

Marcos Bujosa

22 de octubre de 2026

Índice

1. La mecánica en una página	3
1.1. Las tres piezas	3
1.2. Vocabulario	3
1.3. Listas = sorteo; elementos consecutivos = opciones	4
1.4. Mundos, axiomas y la regla de la negación	5
1.5. Partes	5
1.6. Cómo se pegan los fragmentos del enunciado	5
1.7. Un ejemplo completo comentado	5
2. El patrón recomendado: tres huecos y la opción por defecto	7
2.1. Cuatro opciones con mezcla garantizada	7
2.2. Un hueco con una alternativa por mundo	7
2.3. Cuántas variantes	7
3. Preguntas con números: setup, seed e instances	8
3.1. setup	8
3.2. seed	8
3.3. instances	9
3.4. Sortear textos, no solo números	9
3.5. No afirmes nada falso sobre los números: precond con lambda	9
3.6. Cuando la verdad depende de los números: semantica con lambda	10
3.7. Casos especiales solo a veces	10
3.8. Dos trampas de los setup aleatorios	10
4. Qué estilo de pregunta según el contenido	10
5. Redactar cuestiones	11
5.1. Cada cuestión se sostiene sola	11
5.2. Símbolos definidos	12
5.3. El enunciado desnudo; la justificación, en exp	12
5.4. Gemelas: cada verdadera, con su falsa parecida	12
5.5. Afirmaciones que no funcionan como opción	12
5.6. Textos cortos	13
6. Escribir exp	13
7. L^AT_EX y Moodle: trampas conocidas	14
8. Proceso de trabajo	15
8.1. Checklist	15

Esta guía explica **cómo diseñar buenas preguntas** con `qbank`: qué hace el motor con lo que escribes, qué patrón de pregunta conviene adoptar y qué errores de redacción se repiten una y otra vez. No es la referencia de la API (eso es el *Manual*, `Manual.org` o `Manual.pdf`) ni un recorrido por ejemplos (el cuaderno `Tutorial.ipynb`): es lo que hay que tener en la cabeza antes de escribir una pregunta y lo que hay que comprobar después.

Supone que tienes un proyecto creado con `qbank-init` (el `README.org` que `qbank-init` deja en el proyecto explica el flujo de trabajo) y que sabes lanzar `make lint` y `make amc` desde la carpeta del proyecto. Los ejemplos se escriben como listas de Python, que se leen mejor; el JSON equivalente aparece una vez completo en la sección 9 («El mismo ejemplo en JSON»). Todo lo que se dice vale igual para los dos formatos.

1. La mecánica en una página

1.1. Las tres piezas

Una pregunta (`ProblemaTipo`) es una lista plana de componentes de tres tipos:

- **Texto** (una cadena): fragmento fijo del enunciado.
- **Supuesto**: fragmento de enunciado que, además de texto, aporta una **hipótesis lógica** (*semantica*) al resto de la variante.
- **Cuestion**: una afirmación que se ofrece como opción de respuesta. Se marca **correcta** si su *semantica* es consecuencia lógica de las hipótesis acumuladas en esa variante; si no, incorrecta.

Las dos últimas tienen un campo `precond`: la condición que deben cumplir las hipótesis ya aceptadas (o los números del `setup`) para que ese componente pueda aparecer. Si no se cumple, **toda la variante se descarta**: no se muestra al alumno. `precond` no convierte una opción en incorrecta; la quita de en medio.

Las fórmulas se escriben con `calccprop`: `v('A')` es una proposición atómica cuyo nombre eliges tú y solo tiene que ser consistente dentro de la misma pregunta; `-`, `&`, `|`, `>>` y `**` son negación, conjunción, disyunción, implicación y equivalencia; `unoDe(A, B, C)` («exactamente una») y `alguno(A, B, C)` («al menos una») evitan anidar operadores. `True` y `False` valen como fórmulas para afirmaciones siempre ciertas o siempre falsas sea cual sea el mundo.

La pregunta que el motor se hace para cada **Cuestion** es «¿se deduce esto de las hipótesis?», no «¿es verdad en el mundo real?». Esa distinción está detrás de casi todo lo que sigue.

1.2. Vocabulario

Hipótesis lo que la variante da por supuesto: las fórmulas de los **Supuesto** activos. **Consecuencia lógica** de las hipótesis es lo que se sigue necesariamente de ellas (si las hipótesis son ciertas, eso es cierto). Es lo único que el motor comprueba.

Etiqueta cada `v('nombre')`: un nombre que tú eliges para una afirmación («el país es proteccionista», «la función crece»). Solo tiene que ser consistente dentro de la misma pregunta.

Axioma un **Supuesto** sin texto que añade una hipótesis en todas las variantes; casi siempre, que las etiquetas de los mundos son excluyentes (`unoDe`).

Variante cada una de las versiones concretas de la pregunta que ve un alumno: un enunciado y sus opciones, con las marcas de correcta o incorrecta ya calculadas.

Mundo el conjunto de hipótesis activas en una variante; viene determinado por qué **Supuesto** se han sorteado. Dos variantes con los mismos **Supuesto** viven en el mismo mundo aunque muestren opciones distintas.

Hueco cada posición de la pregunta que se convierte en una opción para el alumno: una **Cuestion** suelta, o una lista de **Cuestion** de la que se sortea una.

Parte cada bloque de opciones de una pregunta con varios (ver más abajo). La mayoría de preguntas tienen una sola.

Opción por defecto la opción «Ninguna de las anteriores» que añade `export.last_choice`; es correcta solo si las demás son incorrectas.

Vista profe el PDF de `make amc`: una entrada por mundo (no por variante) con *todas* las cuestiones de cada hueco (no solo la sorteada), su marca y las rechazadas por su precondition. Con `setup`, una entrada por *caso*: cada combinación distinta de mundo y patrón de marcas que aparece al explorar varios sorteos. `nueva_pregunta.py` imprime en texto las primeras entradas de esa vista (las rechazadas con `.`, sin su condición, y sin las explicaciones).

Instancia con `setup` aleatorio, cada ejecución del `setup`; con `export.instances = N` cada variante estructural se genera N veces con números distintos.

1.3. Listas = sorteo; elementos consecutivos = opciones

Cada componente de la lista plana es, o bien un valor suelto, o bien una **lista de dos o más valores del mismo tipo**. En cada variante:

- un valor suelto siempre se incluye;
- de una lista de N alternativas se elige **una sola**.

El número total de variantes es el producto de las longitudes de todas las listas. Y aquí está la consecuencia que rompe la intuición de casi todo el que empieza:

Una lista de varias **Cuestion** **no** significa «estas N afirmaciones se muestran juntas como las N opciones de la pregunta». Significa «en cada variante se sorteará una de estas N ». El alumno ve tantas opciones como **elementos-Cuestion consecutivos** haya en la lista plana (suelos o cada uno como su propia lista de alternativas), más la opción por defecto si `export.last_choice` está activo.

En el esquema siguiente, `S1`, `S2`, `S3` son objetos **Supuesto** y `C1a`, `C1b`, `C2`, ... son objetos **Cuestion**; las cadenas entre comillas son texto fijo:

["Enunciado ",	← cadena: texto fijo, aparece siempre
[S1, S2, S3],	← lista de Supuesto: 3 mundos, se sortea uno
" Entonces:",	← cadena
[C1a, C1b],	← hueco 1, lista de Cuestion: se muestra una de las dos
C2,	← hueco 2, Cuestion suelta: se muestra siempre
[C3a, C3b, C3c]]	← hueco 3, lista de Cuestion: se muestra una de las tres

Variantes: $3 \times 2 \times 1 \times 3 = 18$. Opciones por variante: 3 (+1 con `last_choice`).

Escribir las opciones como **una única lista** de cuatro **Cuestion** produce preguntas de una sola opción; escribirlas todas **sueeltas** muestra siempre las mismas cuatro, sin variedad. `lint` avisa de las dos cosas: de la primera con **pocas-opciones**, de la segunda con **variantes** («solo 1 variantes»).

Una lista de varias **cadenas de texto** también es válida (alternativas de redacción sin lógica asociada), pero se usa poco.

1.4. Mundos, axiomas y la regla de la negación

Una lista de varios **Supuesto** consecutivos son **mundos alternativos**: en cada variante se activa uno, su texto entra en el enunciado y su fórmula en las hipótesis. Es la forma natural de las preguntas «si se cumple A, entonces...; si se cumple B, entonces...»: cada **Cuestion** lleva como **semantica** la etiqueta del mundo (o de los mundos) en que es cierta, y será correcta solo cuando ese sea el mundo sorteado.

Un **Supuesto** suelto con **enunciado** vacío es un **axioma**: añade su fórmula sin añadir texto. Sirve para declarar relaciones entre etiquetas que el motor no conoce. La más habitual: que los mundos son excluyentes, `unoDe(v('A'), v('B'), v('C'))`.

Sin ese axioma, el motor no puede deducir **negaciones**: de la hipótesis `v('crec')` no se sigue `-v('decr')`, porque nada dice que ambas no puedan ser ciertas a la vez. Una **Cuestion** con `semantica=-v('decr')` sería falsa en todos los mundos (lint lo marca como **ERROR negacion**). Dos salidas: formular en positivo (`semantica=v('crec')`) o añadir el axioma `unoDe`. Para una afirmación cierta en varios mundos no hace falta axioma: basta la disyunción de sus etiquetas (`semantica=v('A') | v('B')`).

1.5. Partes

Mientras encadenas elementos-**Cuestion** sin que aparezca ningún texto ni **Supuesto** en medio, todos pertenecen a la **misma parte** (un bloque de opciones). En cuanto, después de una **Cuestion**, vuelve a aparecer texto o un **Supuesto**, se abre una **parte nueva**: un ejercicio con varios bloques de opciones bajo un enunciado común. En Moodle se exporta como una pregunta *cloze* (un bloque de opciones por parte) y en AMC como una única pregunta con varios bloques de casillas, cada uno con su opción por defecto. Las hipótesis se acumulan a lo largo de todas las partes. Para la mayoría de preguntas basta una parte; no fuerces varias si no aportan claridad.

1.6. Cómo se pegan los fragmentos del enunciado

Dos fragmentos consecutivos se concatenan tal cual si alguno de los dos ya aporta separación (empieza o termina en espacio, o está vacío); si ninguno la aporta, `qbank` inserta un espacio. Para controlar la puntuación: pega la coma o el punto al fragmento anterior ("`...,`") y antepón tú el espacio al siguiente (" `entonces`"). Y no pongas la puntuación justo después de un **Supuesto** que pueda estar vacío: con `[Supuesto("", ...), Supuesto("y B$", ...)]` seguido de "`, entonces`" la variante vacía produce «A, entonces». Pon la coma dentro de los **Supuesto** no vacíos («y B,») o mueve el **Supuesto** vacío a otro sitio, y revisa **todas** las variantes renderizadas.

1.7. Un ejemplo completo comentado

```
from qbank import *

p = ProblemaTipo(
    [
        "Sea  $f$  una función ",
        [
            # mundos: en cada variante se sortea uno
            Supuesto("estrictamente creciente", v('crec')),
            Supuesto("estrictamente decreciente", v('decr')),
        ],
        Supuesto("", unoDe(v('crec'), v('decr'))), # axioma: exactamente uno
        " y sean  $a < b$  dos puntos de su dominio. Entonces:",
        [
            # hueco 1: se muestra una de las dos
            Cuestion(" $f(a) < f(b)$ ", v('crec'),
                exp="Si  $f$  crece,  $a < b$  implica  $f(a) < f(b)$ ; si decrece,  $f(a) > f(b)$ ."),
            Cuestion(" $f(a) > f(b)$ ", v('decr'),
                exp="Si  $f$  crece,  $a < b$  implica  $f(a) < f(b)$ ; si decrece,  $f(a) > f(b)$ ."),
```

```

],
Cuestion("$f(a) = f(b)$", False, # hueco 2: se muestra siempre
    exp="Una función estrictamente monótona no repite valores."),
[ # hueco 3
    Cuestion("$f(b) - f(a) > 0$", v('crec'),
        exp="El signo de $f(b)-f(a)$ con $a<b$ es el de la monotonía."),
    Cuestion("$f(b) - f(a) < 0$", v('decr'),
        exp="El signo de $f(b)-f(a)$ con $a<b$ es el de la monotonía."),
],
],
export={"last_choice": True, "cols": 2},
nombre="L-01-Monotonia",
)

```

Hay $2 \times 2 \times 1 \times 2 = 8$ variantes; en todas el alumno ve tres opciones más «Ninguna de las anteriores». Dos de ellas, la 1 y la 6 (el listado no incluye la opción por defecto, que va siempre al final):

1 Sea f una función estrictamente creciente y sean $a < b$ dos puntos de su dominio.
 $\hookrightarrow$ Entonces:
 ✓ $f(a) < f(b)$
 × $f(a) = f(b)$
 ✓ $f(b) - f(a) > 0$

6 Sea f una función estrictamente decreciente y sean $a < b$ dos puntos de su dominio.
 $\hookrightarrow$ Entonces:
 × $f(a) < f(b)$
 × $f(a) = f(b)$
 ✓ $f(b) - f(a) < 0$

Fíjate en que $f(a) > f(b)$ no está escrita como negación de la otra: lleva $v('decr')$ y es correcta solo en ese mundo. Y en que la opción fija $f(a) = f(b)$ es siempre falsa: con `last_choice` eso no es un problema (hay variantes en las que las tres reales son falsas y la opción por defecto pasa a ser la correcta), pero es un distractor que el alumno aprende a descartar; `lint` lo señala como **INFO constante**.

La tabla de mundos resume la lógica de un vistazo y es la forma más rápida de auditarla. Se pide con `make tabla PREGUNTA=<nombre>` (con o sin `.json`), que ejecuta `python -m qbank.lint preguntas/<nombre>.json --tabla`: para esta pregunta, una vez guardada en `preguntas/`, `make tabla PREGUNTA=L-01-Monotonia`; la plantilla de `qbank-init` trae una versión con `setup` de la misma idea: `make tabla PREGUNTA=L-01-FuncionMonotona`. Cada fila es una `Cuestion`, identificada por su posición en la lista plana ([4.1]: componente número 4 y, dentro de él, alternativa número 1, numerando ambos desde 0), cada columna un mundo, y V / F si en ese mundo la cuestión es correcta o incorrecta (- si su precondition la excluye):

Mundos:

M1: estrictamente creciente
 M2: estrictamente decreciente

	M1	M2
[4.0] $f(a) < f(b)$	V	F
[4.1] $f(a) > f(b)$	F	V
[5.0] $f(a) = f(b)$	F	F
[6.0] $f(b) - f(a) > 0$	V	F
[6.1] $f(b) - f(a) < 0$	F	V

(La tabla se calcula con un solo sorteo del `setup`; si los mundos se filtran con preconditiones sobre los números —ver más abajo— solo aparecen los que ese sorteo admite. Por la misma razón,

una `Cuestion` con `semantica` lambda que sale constante en la tabla es solo `INFO constante`: en otros sorteos puede cambiar. Para esas preguntas, la vista profe de `make amc` es la referencia.)

Y `lint` dice exactamente lo que falta (`comp 5[0]` es la misma numeración de la tabla; «semantica literal» significa que la fórmula es el `False` literal, un distractor deliberado):

```
AVISO [variantes] solo 8 variantes (mínimo recomendado 16)
```

```
INFO [constante] falsa en los 2 mundos en que aparece (semantica literal) @ comp 5[0]: $f(a)  
↪ = f(b)$
```

Ocho variantes son pocas; la sección siguiente dice cómo conseguir más sin alargar la pregunta.

2. El patrón recomendado: tres huecos y la opción por defecto

2.1. Cuatro opciones con mezcla garantizada

Cada pregunta (cada parte, si hay varias) debe mostrar siempre el mismo número de opciones, con la garantía de que **al menos una es correcta y al menos una incorrecta** en toda variante (Moodle rechaza una pregunta sin ninguna correcta; un examen en papel con «todas verdaderas» regala la respuesta). Dos formas de lograrlo:

1. **Tres elementos-Cuestion consecutivos** + `"export": {"last_choice": true}`. La cuarta opción («Ninguna de las anteriores» en AMC, «Las demás opciones son falsas» en Moodle) la calcula el motor: es correcta si y solo si las tres reales son incorrectas. Eso garantiza la mezcla sea cual sea la combinación sorteada, sin pensar. Es el patrón por defecto; las preguntas de ejemplo de `qbank-init` lo siguen.
2. **Cuatro elementos-Cuestion** sin `last_choice`, solo si puedes demostrar que, para cualquier combinación de mundo y alternativas, nunca salen las cuatro verdaderas ni las cuatro falsas.

`lint` comprueba la garantía sobre **todas** las variantes (`ERROR mezcla`) con la misma regla que usan los exportadores.

2.2. Un hueco con una alternativa por mundo

El patrón habitual para cada hueco cuando hay mundos: una lista con una `Cuestion` por mundo (con tres mundos A/B/C, tres cuestiones con `semantica v('A')`, `v('B')`, `v('C')`). En cada variante se sortea una, que será correcta solo si coincide con el mundo activo. El número de aciertos entre las tres sorteadas varía de forma natural entre variantes (0, 1, 2 o 3) sin forzarlo a mano.

Cuando una afirmación es cierta en un mundo y falsa en el otro, prefiere **una sola** `Cuestion` con el mismo texto y `semantica=v('mundoA')` a dos redacciones (una en positivo para un mundo, otra en negativo para el otro): es más corta, y garantiza por construcción que la frase tiene sentido sea cual sea el mundo activo. Reserva la lista de alternativas para cuando de verdad hay varias formulaciones distintas que sortear.

Cuando **no** hay alternativa natural para un hueco (una afirmación siempre verdadera o siempre falsa), déjala como `Cuestion` suelta; no inventes alternativas artificiales por rellenar.

2.3. Cuántas variantes

- Sin `setup`: no menos de 16-20 variantes, salvo casos excepcionales.

- Con `setup` aleatorio: al menos 60 en total, ajustando `export.instances` (`lint` calcula el valor con un `INFO setup`). `lint` cuenta variantes, no variantes *distintas*: si el `setup` tiene pocos valores posibles, un `instances` alto repite variantes (ejemplosManual/ej_e_lambda.json, con 60 instancias, da solo 11 preguntas distintas). Mira la vista alumno (`make amc-student`).

La forma más barata de ganar variedad sin alargar ninguna variante concreta es envolver en una lista de **dos o tres redacciones cortas equivalentes** (mismo significado, misma *semantica*) las *Cuestion* que iban sueltas: «está bien definida» / «se puede calcular» / «tiene sentido». Una pregunta con $2 \times 2 \times 1 \times 2 = 8$ variantes pasa a $2 \times 2 \times 3 \times 2 = 24$ sin que ninguna variante sea más larga.

Añadir mundos solo porque sí no es buena idea: varias preguntas centradas, una por concepto, son mejores que una megapregunta con mundos anidados.

3. Preguntas con números: `setup`, `seed` e `instances`

Úsalo cuando el contenido se preste a practicar un cálculo concreto y quieras que cada alumno vea números distintos. Para preguntas conceptuales (definiciones, propiedades, relaciones lógicas) no hace falta `setup`: bastan los mundos.

3.1. `setup`

Es código Python, como cadena, que se ejecuta una vez por variante. Todas las variables que define (salvo las que empiezan por `_`) quedan disponibles para:

- interpolación `@{variable}` en cualquier texto (enunciado, *Supuesto*, *Cuestion* y *exp*);
- lambdas en *semantica* y *precond*, vía `ns['variable']`.

```
SETUP = r"""
import numpy as np
a = int(np.random.randint(1, 10))
b = int(np.random.randint(1, 10))
while b == a:
    b = int(np.random.randint(1, 10))
cuad = (a + b) ** 2
"""
```

Usa `np.random` (no `random`): `seed` siembra `numpy.random` y `random`, pero los scripts de la vista profe (`make amc` y `nueva_pregunta.py`) solo siembran `numpy` en cada sorteo. Calcula en el `setup` todo lo que vaya a aparecer en los textos (la respuesta correcta, los distractores numéricos, cadenas enteras si hace falta): los textos solo interpolan, no calculan. Formatea ahí también los decimales como quieras que se lean (`f"{x:.2f}"`, o la coma decimal como `0{,}5`), porque la interpolación escribe el valor tal cual.

La interpolación de un nombre desconocido **no falla**: deja `@{nombre}` tal cual. `lint` lo detecta (`ERROR interp`), pero revisa igualmente los enunciados generados. Si quieres que el alumno vea una arroba seguida de llaves, escríbela doble: `@@{x}` se imprime `@{x}` y no es un error.

3.2. `seed`

Un entero fijo por pregunta, presente solo si el `setup` usa aleatorios. Garantiza que la variante «1» sea siempre la misma, independientemente de en qué orden se procesen los ficheros del banco. Un convenio cómodo: el número de la lección.

3.3. instances

Cada recorrido de las variantes llama al `setup` una vez por variante; con `export.instances = N` se recorre N veces con semillas distintas. La regla:

```
instances = max(1, ceil(60 / variantes_por_instancia))
```

`lint` imprime el número de variantes por instancia y el `instances` necesario (si los `Supuesto` llevan precondiciones sobre los números, el número de variantes cambia de un sorteo a otro: `lint` usa la media, y el total real puede diferir un poco). Sin `setup` aleatorio, no pongas `instances` (o déjalo en 1).

3.4. Sortear textos, no solo números

El `setup` puede sortear cualquier cosa: nombres, países, décadas, frases. `np.random.choice` elige de una lista de cadenas o de números (convértelo con `str()` o `int()` para que sea un valor normal de Python; con una lista de tuplas o de diccionarios falla, y entonces se sortea la posición: `datos[int(np.random.randint(len(datos)))]`), un diccionario resuelve las concordancias, y las lambdas comparan cadenas igual que números:

```
SETUP = r"""
import numpy as np
pais = str(np.random.choice(["España", "Francia", "Portugal"]))
gentilicio = {"España": "española", "Francia": "francesa", "Portugal": "portuguesa"}[pais]
decada = int(np.random.choice([1880, 1890, 1900]))
"""

PREGUNTA = ProblemaTipo(
    ["Suponga que en la década de @{decada} el gobierno de @{pais} ",
     [Supuesto(r"establece un arancel del 20 \% sobre las importaciones.", v('prot')),
      Supuesto("suprime todos los aranceles sobre las importaciones.", v('libre'))],
     Supuesto("", unoDe(v('prot'), v('libre'))),
     " A corto plazo:",
     [Cuestion("los bienes importados se encarecen", v('prot')),
      Cuestion("los bienes importados se abaratan", v('libre'))],
     [Cuestion("la industria @{gentilicio} queda protegida de la competencia exterior",
               ↪ v('prot')),
      Cuestion("la industria @{gentilicio} queda más expuesta a la competencia exterior",
               ↪ v('libre'))],
     Cuestion("el país es @{pais}", "lambda ns: ns['pais'] == 'Francia')], # ejemplo de
     ↪ lambda sobre texto
    setup=SETUP, seed=3, export={"last_choice": True, "cols": 1, "instances": 8},
    nombre="L-03-PoliticaArancelaria")
```

La pregunta `L-03-PoliticaArancelaria` de la plantilla es la versión completa de este ejemplo (sin la última cuestión, que solo ilustra la lambda). Dos detalles de prosa: el % lleva barra (\%) y la cadena que lo contiene es cruda (`r"..."`), ver la sección 7; y con opciones de una línea o más conviene `cols = 1`.

3.5. No afirmes nada falso sobre los números: **precond** con lambda

El motor razona de forma proposicional y **no conoce** los valores. Si ofreces los supuestos « $a > b$ » y « $a < b$ » sin más, generará variantes en las que el texto dice $a > b$ con $a = 3$ y $b = 7$. La solución es una precondición que solo deje pasar la variante cuando lo que afirma es cierto para los números sorteados:

```
[
    Supuesto("$a > b$", True, precondition="lambda ns: ns['a'] > ns['b']"),
    Supuesto("$a < b$", True, precondition="lambda ns: ns['a'] < ns['b']"),
],
```

Las variantes cuyo **Supuesto** no supera la precondition se descartan sin más (no aparecen ni en la vista profe). Una **Cuestion** con precondition también se puede filtrar así (`precond="lambda ns: ns['k'] != 2"`): en la vista alumno la variante se descarta igual; en la vista profe la cuestión aparece al final, en «Exclusiones», con su condición.

Las lambdas se escriben **como cadena** también en Python (`precond="lambda ns: ..."`): así la pregunta puede guardarse en JSON. Una lambda escrita como función Python no se puede guardar.

3.6. Cuando la verdad depende de los números: **semantica con lambda**

Si lo que decide la corrección es un cálculo (no un mundo), la semántica es una lambda que devuelve True o False:

```
Cuestion("$a^2 > b^2$", "lambda ns: ns['a'] ** 2 > ns['b'] ** 2"),
```

Una lambda también puede devolver una **fórmula** y así inyectar en las hipótesis el estado de los números (`lambda ns: v('gt') if ns['x'] > 3 else -v('gt')` en la semántica de un **Supuesto** axioma); `ejemplosManual/ej_e_lambda.json` (el ejemplo E del cuaderno `Tutorial.ipynb`) lo desarrolla. Como `lint` no puede mirar dentro de una lambda, cuando un **Supuesto** inyecta fórmulas así no comprueba las etiquetas huérfanas y lo dice con un INFO.

3.7. Casos especiales solo a veces

Si quieres que un caso degenerado (un vector nulo, dos números iguales) aparezca ocasionalmente —ni nunca ni en la mitad de las variantes—, no lo controles con una lista de alternativas (eso daría el mismo peso a cada una): sorteas una probabilidad dentro del **setup**, con ramas excluyentes, y condiciona la **semantica** con una lambda. Esto no añade factores a la combinatoria. Comprueba el porcentaje real contando sobre `p.por_partes(instances=...)` antes de aceptar la pregunta.

3.8. Dos trampas de los **setup** aleatorios

- **Condiciones incompletas.** Si una semántica depende de una propiedad estructural (independencia lineal, coprimidad...), compruébala entera en el **setup** (rango, máximo común divisor), no solo el caso obvio («¿es nulo?»). Y pruébalo con muchas semillas distintas, no solo con la *seed* del banco: un caso raro puede no aparecer con esa semilla y aun así ser un fallo real cuando cambien *instances* o el rango de valores.
- **Distractores que coinciden con la respuesta.** Si el **setup** calcula un valor falso para una opción (`$(a+b)^2 = @{cuad_falso}$`), asegúrate de que nunca coincide con el correcto; si puede, re-sorteas hasta que difieran. En general, excluye en el enunciado o en el **setup** los casos degenerados en que un distractor pasa a ser cierto (un coeficiente nulo, dos valores iguales, un factor igual a ± 1 , un vector nulo).

4. Qué estilo de pregunta según el contenido

- **Definiciones y propiedades:** pregunta puramente lógica, sin **setup**, a menudo sin mundos siquiera: un enunciado y tres huecos con afirmaciones siempre verdaderas o siempre falsas (y la opción por defecto):

```

ProblemaTipo(
  ["Sea $$ un número primo mayor que 2. Entonces:",
   [Cuestion("$p$ es impar", True),
    Cuestion("$p$ es par", False)],
   [Cuestion("$p$ tiene exactamente dos divisores positivos", True),
    Cuestion("$p$ tiene exactamente un divisor positivo", False)],
   [Cuestion("$p + 1$ es par", True),
    Cuestion("$p + 1$ es primo", False)]],
  export={"last_choice": True}, nombre="L-01-Primos")

```

(Solo $2 \times 2 \times 2 = 8$ variantes: para un banco real añade redacciones alternativas o un cuarto hueco; aquí interesa la forma.)

- **Resultados condicionales** («si se cumple tal hipótesis, entonces...»): mundos alternativos, uno por hipótesis relevante, y **Cuestion** cuya **semantica** depende del mundo. Úsalos cuando el contenido hace una distinción real; no los fuerces cuando no la hay (una pregunta con varias cuestiones independientes y sin **Supuesto** es perfectamente legítima).
- **Cálculo numérico**: setup con `np.random` e interpolación.
- **Una pregunta por concepto clave**. No fijas de antemano cuántas preguntas saldrán de un tema: cuenta los resultados genuinamente distintos y deja que ese número decida. Forzar más produce preguntas redundantes.
- Si un concepto de un tema anterior reaparece generalizado, reutilizar el patrón de aquella pregunta es un refuerzo didáctico legítimo: antes de diseñar una pregunta nueva, mira si ya existe una sobre el concepto que se generaliza.
- **No evalúes contenido de temas futuros**. Que el material haga un «guiño» a algo que se demuestra más adelante no lo convierte en evaluable con las herramientas de este tema. Si quieres mencionarlo, hazlo en el enunciado o en **exp** como contexto, nunca como algo que juzgar verdadero o falso. Expresiones como «volveremos sobre esto» o «más adelante veremos» son la señal de alarma.

5. Redactar cuestiones

5.1. Cada cuestión se sostiene sola

Para **cada** **Cuestion** candidata (no solo las que salen en la primera variante), concatena mentalmente el enunciado fijo de esa variante (incluido el **Supuesto** activo) con el texto de esa cuestión **sola**. El resultado debe ser un texto con sentido pleno, cuya verdad o falsedad quede determinada por lo que dice esa concatenación, nunca por lo que «podría» haber salido en otro hueco. Como cada hueco se sortea de forma independiente, no hay garantía de que ninguna otra cuestión en concreto acompañe a la que revisas. Esto descarta:

- «esto ocurre porque...», «esa igualdad se cumple porque...», con un «esto» que presupone una afirmación de otro hueco. Si quieres preguntar por qué se cumple X , repite X entero dentro de la cuestión;
- cualquier **exp** que remita a «el principio anterior» o «la misma idea» señalando a otra cuestión que puede no haberse mostrado.

Dos alternativas del **mismo** hueco nunca se muestran juntas; las de huecos distintos, solo a veces. Y una alternativa que es la negación literal de otra de **otro** hueco puede salir junto a ella y delatarla.

5.2. Símbolos definidos

Todo símbolo que aparezca en una **Cuestión** ($\theta, \alpha, n, U \dots$) debe estar definido en el enunciado de esa variante o ser notación fija del curso que tu proyecto declare como conocida. Si el enunciado fija objetos **concretos** («sean $x = (1, 2)$ e $y = (3, 1)$ ») y una **Cuestión** enuncia una identidad **general**, usa letras distintas en la identidad (u, v): con los mismos símbolos, «solo se cumple si $x \perp y$ » tiene doble lectura cuando esos x, y concretos resultan ser ortogonales.

Cuidado también con una **Cuestión** que nombra un símbolo propio de un mundo concreto: si su **precond** es **True** (lo normal), puede sortearse bajo el mundo contrario, donde ese símbolo nunca se introdujo. Prefiere una redacción genérica («el vector que se suma») o restringe la alternativa a su mundo con **precond**.

Si varios mundos introducen parámetros análogos, **usa el mismo símbolo en todos los Supuesto**: cada fórmula con ese símbolo es entonces una única **Cuestión** válida en cualquier mundo, con **semantica** igual a la disyunción de los mundos donde es cierta, en vez de repetir la misma fórmula varias veces con **precond** distintos.

5.3. El enunciado desnudo; la justificación, en exp

No escribas « X se cumple porque Y » ni « X , por el teorema Z » dentro de una cuestión: pon « X » y mueve el porqué a **exp**. Además de acortar la pregunta, evita el patrón más fácil de aprender que existe: **la opción larga es la verdadera**. Es real y medible: en un banco revisado, las opciones verdaderas medían de media entre 1,7 y 2 veces más que las falsas porque la verdadera llevaba su «porque...» y la falsa iba desnuda. Comprobación mecánica antes de aceptar una pregunta: media de caracteres de las opciones verdaderas frente a las falsas; si difieren en más de un 30 %, quita las justificaciones de las verdaderas o da a las falsas una justificación plausible pero errónea.

5.4. Gemelas: cada verdadera, con su falsa parecida

La estrategia más eficaz para que una pregunta discrimine: **cada vez que escribas una cuestión verdadera, busca un distractor falso y parecido** en el mismo hueco. Es como preguntar lo mismo dos veces, una bien y otra mal: distingue a quien sabe por qué algo está bien y sabe identificar el error. Y evita los distractores de relleno sin sentido: cada falsa nace de una afirmación con sentido. Gemelas típicas: papeles intercambiados (numerador/denominador), el signo o la desigualdad al revés, la fórmula con el divisor equivocado, el valor numérico con los datos cruzados.

Una afirmación numérica **siempre cierta y sin gemela** ($(a+b)^2 = \text{@{cuad}}\$$ sin su $(a+b)^2 = \text{@{cuad_falso}}\$$) se marca sin comprobar nada.

5.5. Afirmaciones que no funcionan como opción

- **Igualdades con ambos lados ya calculados** (« $20 = 4$ »): se responden por inspección visual. Deja la identidad simbólica o pregunta por el valor de una sola cantidad.
- **Ejemplos en dimensión muy baja** para ilustrar un cálculo general: con dos datos, cada uno dista de la media lo mismo que el otro, un artefacto que no generaliza a tres. Reserva la dimensión baja para lo que solo tiene sentido ahí (ángulos, ortogonalidad visual).
- **«Generalmente», «puede cambiar»**: busca la reformulación más fuerte y determinista que capture lo mismo («deja de ser exactamente el mismo» en vez de «generalmente cambia»).
- **Afirmaciones compuestas** (« n es par y $n^2 + 1$ es par», mitad verdadera y mitad falsa): parte en dos o quédate con la mitad que evalúa el concepto.

- **Cuestiones sobre el modo de exponer** («no hace falta una demostración nueva para esto»): no son una propiedad verificable.
- **Repetir una cuestión de otro hueco con relleno distinto**: no es una alternativa nueva, y hace que unos alumnos vean menos alternativas realmente distintas que otros.
- **Patrones fijos que se aprenden** («el primer caso siempre falla», «la tercera opción siempre es la buena»): si el **setup** lo produce, rómpelo (intercambio aleatorio de papeles).

5.6. Textos cortos

Cuanto más corta sea cada pregunta, más caben por página. Antes de dar un **Supuesto** o una **Cuestión** por buenos, pregúntate si puedes decir lo mismo con menos palabras: fuera incisos, ejemplos redundantes con el enunciado, explicaciones que ya van en **exp**, la palabra «exactamente» cuando no aporta (« $\Leftrightarrow$ » vale por «ocurre exactamente cuando»). El límite es que la pregunta se siga entendiendo sin ambigüedad.

Dos columnas (`export.cols = 2`) ocupan menos papel que una en casi todas las preguntas de opciones cortas; con opciones de una línea o más (lo habitual en prosa) usa `cols = 1`, y mira el PDF antes de decidir. Mide con la maquetación real del examen, no con el catálogo de revisión (que imprime el nombre de la pregunta en el encabezado).

6. Escribir exp

exp es lo que Moodle muestra **después** de responder; el alumno la lee junto a la opción, en la variante concreta que le tocó. Reglas:

- **Una razón para cada mundo alcanzable.** Si la **Cuestión** puede aparecer bajo varios mundos (su **precond** no la restringe a uno), **exp** debe dar, para cada uno, una razón concreta anclada en la hipótesis de **ese** mundo, tanto si ahí es verdadera como si es falsa. No basta con explicar el mundo «bueno». Si la razón es genuinamente la misma en varios mundos (un argumento dimensional, por ejemplo), basta una frase que los cubra; el fallo a vigilar es cubrir solo el subconjunto donde la explicación resulta más obvia y olvidar el resto.
- **Sin jerga del diseño.** Nunca «en el mundo $\lambda > 0$ », «en los demás mundos alcanzables», ni las etiquetas de $v(\dots)$. El alumno solo ve la hipótesis escrita en su enunciado: razona con ella («si f es creciente, ...; si es decreciente, ...»).
- **Sin desvelar el diseño global.** Nunca «esta opción es siempre falsa» o «en ninguno de los casos contemplados es cierto», aunque sea verdad: le permite memorizar «esta frase, descártala» en cualquier variante futura, justo lo contrario de lo que se busca al multiplicar variantes.
- **Autocontenida.** Nada de «es la misma identidad de antes», «el valor anterior al cuadrado», «como en el ejemplo de arriba»: la otra alternativa estaba en el mismo hueco y nunca se muestra a la vez. Repite lo que haga falta (la fórmula, el ejemplo).
- **Una razón, no un nombre.** «Es la fórmula del binomio» no explica nada; el porqué es justo lo que se quiere fijar. Y «nada obliga a...» donde la afirmación es rotundamente falsa debe sustituirse por la razón concreta.
- **Compruébala como una demostración.** « $a \neq 0$ y $b \neq 0$, así que $a + b \neq 0$ » es un razonamiento falso ($a = -b$). Cada **exp** es una afirmación matemática más que puede estar mal.

- Con `setup`, interpola los valores concretos (`@{a}`) en vez de quedarte en lo genérico; una sola `exp` con `@{...}` sirve para todas las variantes. Si la explicación depende de una condición numérica, calcula el texto en el `setup` como variable y pon `exp="@{texto_exp}"`; no escribas `exp` como `lambda` en JSON (llegaría literal al alumno).
- Si una `exp` no aporta nada más allá de lo obvio, déjala vacía.

Revisa también las `Cuestion` con `semantica False` a secas: si hay mundos, comprueba en la tabla de mundos que esa afirmación no resulte accidentalmente cierta en alguno (dos cantidades que en general son distintas coinciden en un mundo degenerado). Si ocurre, cambia la semántica a la disyunción de los mundos en que es cierta, o excluye ese mundo con `precond`.

7. L^AT_EX y Moodle: trampas conocidas

Los textos (enunciados, cuestiones y explicaciones) son L^AT_EX, aunque no lleven ni una fórmula. En español puedes teclear tal cual á é í ó ú ñ ü ¿ ; (también en mayúscula) y las comillas tipográficas “ ”; lo que sigue, no. `lint` lo detecta (`texto`; el % sin barra es ERROR, el resto AVISO) y lo que no detecta lo detecta Moodle, que es más caro.

- **El símbolo %:** en L^AT_EX empieza un comentario. Escrito tal cual («un 25 % del valor»), se pierde el resto de la línea, el PDF de AMC sale vacío o roto y Moodle no compila. Escribe siempre `\%` (un 25 % del valor); en Python, dentro de una cadena cruda (`r"..."`) o como `\\%`. `lint` lo busca también en el texto ya interpolado, así que detecta un % que venga de una variable del `setup`. Y si va junto a una fórmula, **fuera de los dólares:** al `$5\,\%`, nunca al `$5\,\%$`, porque el paquete `moodle` deja dentro de la fórmula un % desnudo que MathJax toma por comentario (en papel se ve bien; solo se nota en Moodle). `\,` es un espacio fino, el que va entre la cifra y el símbolo.
- **Comillas:** ni las angulares «» (no existen en la codificación que usa el paquete `moodle`: la compilación falla) ni las rectas "..." (salen como "...", cerradas por los dos lados). Escribe ``así``: dos acentos graves para abrir y dos apóstrofes para cerrar.
- **Grados:** `$180^\circ`, nunca `°` (en modo matemático deja a `pdflatex` colgado sin error).
- Fórmulas con `$$$` y `$$$`; nunca `\(...\)` ni `\[...\]` directamente (`qbank` convierte `$$$` en `\(...\)` al exportar a AMC). Separa dos expresiones en línea contiguas con un espacio (`$A$ $B$`, no `$A$B$`).
- **Tablas con array dentro de \$\$\$**, con `\text{}` en las cabeceras, nunca `tabular`: el paquete `moodle` no lo convierte y el alumno vería el código.
- En Python, escribe los textos con L^AT_EX como **cadena cruda** (`r"$\mathcal{A}$"`) o dobla las barras; en JSON, cada `\` es `\\`.
- No cites números de lección u otras referencias al material en el texto de la pregunta: las preguntas deben ser autocontenidas.
- Comprueba cada pregunta nueva en los dos destinos: `make amc` (el PDF) y `make moodle` (el XML, y a ser posible una vista previa en Moodle). Si `make` dice «ERROR en <pregunta>», la causa está en el `.log` que indica: busca la primera línea que empieza por !.

8. Proceso de trabajo

1. Lee el material completo e identifica los resultados que merezca la pena evaluar. Cuenta cuántos son; ese es el número de preguntas.
2. Para cada uno decide el estilo (mundos, `setup` o cuestiones independientes) y diseña sobre papel: qué `Supuesto`, qué huecos, qué `semantica` y `precond`, si hace falta `setup`.
3. Escribe la pregunta (`nueva_pregunta.py` o el JSON) y pásale `lint`. Un `ERROR` es una pregunta rota; un `AVISO` hay que leerlo y decidir.
4. Genera la vista `profe` (`make amc`) y repasa **todas** sus entradas, no la primera: enunciados bien formados, número de opciones esperado, marcas correctas en cada mundo, varias semillas si hay `setup`. La vista `profe` imprime cada `exp` en cursiva junto a su cuestión. Para leer con calma, `make lectura` produce un documento por lección con un ejemplo por mundo, las alternativas agrupadas por hueco con su V/F y su `exp`, y las excluidas con su motivo: es el formato para el paso 7.
5. Pasa el checklist de abajo cuestión por cuestión.
6. Si algo falla, corrige y vuelve al paso 3; no lo dejes para después.
7. Antes de publicar, que la lea **otra persona** sin el contexto de quien la escribió, contra el material: la segunda mirada encuentra fallos de fondo que ni `lint` ni el autor ven.

8.1. Checklist

- ☐ ¿Cada elemento suelto va sin envolver en una lista de un elemento, y ninguna lista mezcla tipos?
- ☐ ¿Cada hueco es un elemento-Cuestion (suelto o lista de alternativas), y hay tres huecos consecutivos + `last_choice` (o cuatro con garantía demostrada)?
- ☐ ¿Coincide `nombre` con el nombre del fichero, sin `:` ni espacios?
- ☐ ¿Usa el `setup np.random` (no `random`)? ¿Lleva `seed` solo si el `setup` sortea, y da `instances` ≥ 60 variantes, sin demasiadas repetidas en la vista `alumno`? ¿Sin `setup`, hay ≥ 16 ?
- ☐ ¿Las negaciones tienen su axioma `unoDe`, o están formuladas en positivo?
- ☐ ¿Cada cuestión, sola con el enunciado de su variante, tiene sentido pleno y verdad inequívoca? ¿Nada remite a otra alternativa?
- ☐ ¿Todo símbolo está definido, y las identidades generales usan letras distintas de los objetos concretos?
- ☐ ¿El enunciado afirma solo lo que el `precond` garantiza sobre los números?
- ☐ ¿Cada verdadera numérica tiene su gemela falsa? ¿Las opciones verdaderas y falsas miden parecido?
- ☐ ¿Los casos degenerados en que un distractor se vuelve cierto están excluidos?
- ☐ ¿Cada `exp` justifica cada mundo alcanzable, sin jerga del diseño, sin desvelar el diseño global y sin referencias a otras alternativas?
- ☐ ¿Todo% lleva barra (`\%`) y va fuera de los dólares? ¿Sin `°`, sin «», sin comillas rectas? ¿Fórmulas con `$` y tablas con `array`?

- ☐ En prosa: ¿cada afirmación es claramente cierta en su mundo y claramente falsa en los otros, sin «depende»? ¿La hipótesis del Supuesto basta para decidirla, sin suponer conocimientos que no están en el enunciado?
- ☐ ¿Has repasado todas las entradas de la vista profe (o de la vista de lectura) y, con `setup`, varias semillas?
- ☐ ¿Has pasado `make patrones (lint --patrones)` y leído sus avisos? ¿Podría un alumno acertar sin leer el enunciado, por la longitud de la frase, una palabra, una cifra o porque la gemela falsa es siempre la mayor?

Si alguna casilla falla, la pregunta no está lista.

9. El mismo ejemplo en JSON

Es lo que escribe `save_problema` (o `nueva_pregunta.py --guardar`) para el ejemplo de la primera sección, y lo que leen `lint` y los scripts de `make`. Las fórmulas y las lambdas van como cadenas; las barras de L^AT_EX se doblan.

```
{
  "version": "1",
  "tipo": "ProblemaTipo",
  "nombre": "L-01-Monotonia",
  "setup": null,
  "componentes": [
    "Sea  $f$  una función ",
    [
      {"tipo": "Supuesto", "enunciado": "estrictamente creciente",
       "semantica": "v('crec')", "precond": "True"},
      {"tipo": "Supuesto", "enunciado": "estrictamente decreciente",
       "semantica": "v('decr')", "precond": "True"}
    ],
    {"tipo": "Supuesto", "enunciado": "",
     "semantica": "unoDe(v('crec'), v('decr'))", "precond": "True"},
    " y sean  $a < b$  dos puntos de su dominio. Entonces:",
    [
      {"tipo": "Cuestion", "enunciado": " $f(a) < f(b)$ ",
       "semantica": "v('crec')", "precond": "True",
       "exp": "Si  $f$  crece,  $a < b$  implica  $f(a) < f(b)$ ; si decrece,  $f(a) > f(b)$ ."},
      {"tipo": "Cuestion", "enunciado": " $f(a) > f(b)$ ",
       "semantica": "v('decr')", "precond": "True",
       "exp": "Si  $f$  crece,  $a < b$  implica  $f(a) < f(b)$ ; si decrece,  $f(a) > f(b)$ ."}
    ],
    {"tipo": "Cuestion", "enunciado": " $f(a) = f(b)$ ",
     "semantica": "False", "precond": "True",
     "exp": "Una función estrictamente monótona no repite valores."},
    [
      {"tipo": "Cuestion", "enunciado": " $f(b) - f(a) > 0$ ",
       "semantica": "v('crec')", "precond": "True",
       "exp": "El signo de  $f(b)-f(a)$  con  $a < b$  es el de la monotonía."},
      {"tipo": "Cuestion", "enunciado": " $f(b) - f(a) < 0$ ",
       "semantica": "v('decr')", "precond": "True",
       "exp": "El signo de  $f(b)-f(a)$  con  $a < b$  es el de la monotonía."}
    ]
  ],
  "export": {"last_choice": true, "cols": 2}
}
```

El orden de las claves da igual. Con `setup`, el campo es una cadena con el código y los saltos de línea como `\n`; `seed` va al nivel superior (`save_problema` lo omite cuando no hay; escribir `"seed": null` es equivalente); `export.instances` junto a `last_choice` y `cols`. Puedes añadir claves propias de nivel superior (`"_nivel": "M", "_tema": "..."`): `qbank` las ignora, así que sirven para tus metadatos sin romper nada.