

Índice general

I	Introducción	1
1.	Qué puede hacer qbank	3
2.	Instalación rápida (para quien ya conoce Python)	4
II	Instalación paso a paso	5
3.	Requisitos previos	6
4.	Qué es un entorno virtual y por qué usarlo	7
5.	Crear y activar un entorno virtual	8
6.	Instalar calcprop y calcprop-qbank	9
7.	Nota para usuarios de Windows	10
III	Conceptos fundamentales	11
8.	Variables proposicionales con calcprop	12
9.	Supuesto — una hipótesis del enunciado	13
10.	Cuestion — un ítem de respuesta	14
11.	ProblemaTipo — un generador de variantes	15
12.	ProblemaVF — preguntas de verdadero/falso	17
13.	Preguntas paramétricas con setup	18
13.1.	Sin setup: combinatoria pura	18
13.2.	Con setup y valores aleatorios	18
13.3.	Con setup y valores fijos	19
14.	Guardar y cargar problemas en formato JSON	20
14.1.	De lista de listas a JSON	20
14.2.	De JSON a lista de listas	21
14.3.	Con setup paramétrico	22
14.4.	Limitaciones	23
15.	Exportar la lista de listas a un fichero .py editable	24
IV	Exportar a AMC (\LaTeX)	25
16.	Funciones disponibles	27

17.Firma común	28
18.Ejemplo AMC básico	29
19.Ejemplo AMC multicolumna con «Ninguna de las anteriores»	30
20.Integración en el documento AMC	31
V Exportar a Moodle (vía L^AT_EX)	32
21.Flujo de trabajo	34
22.Funciones disponibles	35
23.Firma común	36
24.Ejemplo Moodle completo	37
25.Ejemplo con ProblemaVF para Moodle	38
VI Preguntas multi-parte (ProblemaMultiParte)	39
26.Ejemplo básico	41
27.Exportar a AMC	42
28.Exportar a Moodle (cloze)	44
29.Persistencia JSON	45
30.Exportar a código Python editable	46
VII Editor visual en Jupyter (ProblemaTipoEditor)	47
31.Instalación	49
32.Uso básico	50
33.Descripción del formulario	51
34.Ejemplo sin setup	52
35.Ejemplo con setup paramétrico	53
36.Obtener el problema desde el editor	54
37.Flujo completo: editor → JSON → AMC y Moodle	55
37.1. Paso 1 — Diseñar y guardar desde el editor	55
37.2. Paso 2 — Cargar el JSON	55
37.3. Paso 3a — Exportar a AMC	55
37.4. Paso 3b — Exportar a Moodle	56
37.5. Paso 4 — Generar múltiples instancias numéricas (problemas con <code>setup</code>)	56
VIII Referencia rápida de la API	58
38.Clases	59
39.Funciones JSON	60

40.Funciones AMC	61
41.Funciones Moodle	62
42.Operadores de calcprop (disponibles tras <code>from qbank import *</code>)	63
IX Preguntas frecuentes	64
43.¿Por qué algunas variantes no se generan?	65
44.¿Cómo incluyo texto \LaTeX en los enunciados?	66
45.¿Para qué sirve <code>codchar</code> ?	67
46.¿Cómo genero un diccionario de preguntas de varias secciones?	68

Manual de uso de **qbank**

Marcos Bujosa

10 de junio de 2026

Parte I

Introducción

`qbank` es una librería Python para generar bancos de preguntas de opción múltiple cuyas respuestas correctas se determinan automáticamente mediante cálculo proposicional.

A partir de una lista de supuestos y cuestiones definidos por el usuario, `qbank` genera todas las variantes posibles de una pregunta combinando los elementos alternativos. Para cada variante calcula cuáles respuestas son correctas y cuáles son falsas, y exporta el resultado a los formatos \LaTeX /AMC o Moodle.

La librería se apoya en [calcprop](#), que proporciona el motor de cálculo proposicional.

Capítulo 1

Qué puede hacer qbank

- Definir preguntas con supuestos e ítems intercambiables.
- Calcular automáticamente la corrección de cada ítem según las hipótesis activas en cada variante.
- Exportar las variantes generadas a:
 - Ficheros `.tex` compatibles con [AMC](#) (Auto Multiple Choice).
 - Ficheros `.tex` compilables con el paquete `LATEX moodle`, que produce un `.xml` importable en Moodle.

Capítulo 2

Instalación rápida (para quien ya conoce Python)

```
pip install calcprop calcprop-qbank
```

Si no estás familiarizado con Python o los entornos virtuales, lee la sección siguiente.

Parte II

Instalación paso a paso

Capítulo 3

Requisitos previos

qbank requiere Python 3.9 o superior. Comprueba si ya lo tienes instalado:

Sistema	Comando en terminal
Linux	<code>python3 --version</code>
macOS	<code>python3 --version</code>
Windows	<code>py --version</code>

Si el comando devuelve algo como `Python 3.11.2` estás listo. Si no, descarga Python desde python.org e instálalo (en Windows marca la opción *Add Python to PATH*).

Capítulo 4

Qué es un entorno virtual y por qué usarlo

Un entorno virtual es una carpeta que contiene una copia aislada de Python con sus propios paquetes instalados, independiente del resto del sistema. Esto evita conflictos entre proyectos que requieren versiones distintas de las mismas librerías.

Trabajar siempre dentro de un entorno virtual es una buena práctica, aunque no es estrictamente obligatorio para un uso sencillo de `qbank`.

Capítulo 5

Crear y activar un entorno virtual

Abre una terminal (en Windows: *Símbolo del sistema* o *PowerShell*) y sitúate en el directorio donde quieres trabajar, por ejemplo:

```
mkdir mis_cuestionarios
cd mis_cuestionarios
```

Crea el entorno virtual (llámalo como quieras; aquí usamos `.venv`):

Sistema	Comando
Linux / macOS	<code>python3 -m venv .venv</code>
Windows (cmd)	<code>py -m venv .venv</code>
Windows (PS)	<code>py -m venv .venv</code>

Actívalo:

Sistema	Comando
Linux / macOS	<code>source .venv/bin/activate</code>
Windows (cmd)	<code>.venv\Scripts\activate.bat</code>
Windows (PowerShell)	<code>.venv\Scripts\Activate.ps1</code>

Cuando el entorno está activo, verás su nombre entre paréntesis al comienzo del prompt, por ejemplo (`.venv`). A partir de ese momento, `pip` y `python` apuntan al entorno virtual y no al sistema.

Capítulo 6

Instalar calcprop y calcprop-qbank

Con el entorno activo:

```
pip install calcprop calcprop-qbank
```

Verifica la instalación:

```
python -c "from qbank import *; print('qbank OK')"
```

Si ves qbank OK todo está en orden.

Capítulo 7

Nota para usuarios de Windows

En Windows, dentro del entorno virtual, usa `python` (no `python3`). Si el primer `py -m venv .venv` falla con un error de permisos en PowerShell, ejecuta antes:

```
Set-ExecutionPolicy -Scope CurrentUser RemoteSigned
```

Parte III

Conceptos fundamentales

Capítulo 8

Variables proposicionales con `calccprop`

`qbank` importa automáticamente `calccprop`, así que al escribir `from qbank import *` tienes acceso al motor lógico. Las variables proposicionales se crean con `v()`:

```
from qbank import *  
A = v('A') # variable proposicional A
```

Las fórmulas se construyen combinando variables con los operadores de Python:

Operador Python	Conectiva lógica	Ejemplo
<code>-v('A')</code>	Negación ($\neg A$)	<code>-v('A')</code>
<code>v('A') & v('B')</code>	Conjunción ($A \wedge B$)	<code>v('A') & v('B')</code>
<code>v('A') v('B')</code>	Disyunción ($A \vee B$)	<code>v('A') v('B')</code>
<code>v('A') >> v('B')</code>	Implicación ($A \implies B$)	<code>v('A') >> v('B')</code>
<code>v('A') ** v('B')</code>	Bicondicional ($A \iff B$)	<code>v('A') ** v('B')</code>
<code>alguno(v('A'), v('B'), v('C'))</code>	Alguna verdadera	<code>alguno(v('A'), v('B'), v('C'))</code>
<code>unoDe(v('A'), v('B'), v('C'))</code>	Exactamente una verdadera	<code>unoDe(v('A'), v('B'), v('C'))</code>

La semántica también acepta `True` (siempre verdadero) o `False` (siempre falso). `alguno` y `unoDe` aceptan cualquier número de proposiciones: `alguno(P, Q, R)` es verdadera si alguna de la lista lo es, y `unoDe(P, Q, R)` es verdadera si lo es exactamente una.

La función `test(formula, hipotesis)` comprueba si una fórmula es consecuencia lógica de una lista de fórmulas hipótesis:

```
test(v('C'), [v('A'), v('A') >> v('C')]) # True (modus ponens)  
test(v('C'), [v('B')]) # False (B no implica C)  
test(True, []) # True  
test(False, []) # False
```

No necesitas llamar a `test` directamente: `qbank` lo invoca internamente al generar las variantes.

Capítulo 9

Supuesto — una hipótesis del enunciado

Un **Supuesto** es una afirmación que se añade al enunciado del problema y cuya semántica se incorpora como hipótesis para evaluar las cuestiones que le siguen.

```
Supuesto(enunciado, semantica, precond=True)
```

Parámetro	Tipo	Descripción
enunciado	cadena	Texto \LaTeX del supuesto, tal como aparecerá impreso.
semantica	fórmula	Fórmula proposicional que el supuesto afirma verdadera.
precond	fórmula/bool	El supuesto solo se activa si esta precondición se cumple bajo las hipótesis ya acumuladas. Por defecto <code>True</code> (siempre activo).

Ejemplo:

```
Supuesto('$\mathcal{A}$ es verdadero. ', v('A'))  
Supuesto('$\mathcal{A} \Rightarrow \mathcal{C}$.', v('A') >> v('C'))
```

Si la precondición de un supuesto no se cumple en una variante concreta, esa variante se descarta silenciosamente.

Capítulo 10

Question — un ítem de respuesta

Una `Question` es una posible respuesta en el cuestionario. `qbank` determina automáticamente si es verdadera o falsa según las hipótesis activas en cada variante.

```
Question(enunciado, semantica, precond=True, exp="")
```

Parámetro	Tipo	Descripción
<code>enunciado</code>	cadena	Texto \LaTeX del ítem de respuesta.
<code>semantica</code>	fórmula/bool	Fórmula cuya veracidad determina si el ítem es correcto.
<code>precond</code>	fórmula/bool	La cuestión solo aparece en la pregunta si esta condición se satisface. Por defecto <code>True</code> .
<code>exp</code>	cadena	Explicación/retroalimentación (feedback) en \LaTeX . Opcional.

Ejemplos:

```
# Ítem siempre incluido; correcto cuando C es verdadero
Question("Entonces  $\mathcal{C}$  es verdadero", v("C"))

# Ítem siempre falso (False como semántica)
Question("Esta afirmación es siempre falsa", False)

# Ítem que solo aparece cuando A es verdadero; correcto cuando también B lo es
Question(" $\mathcal{B}$  es consecuencia de  $\mathcal{A}$ ", v("B"), precond=v("A"))

# Ítem con retroalimentación para el alumno
Question("Las columnas son linealmente independientes", -v("Mcoli"),
  exp=r"Solo cuando  $\text{rg}(\mathbf{X}) = k$ ")
```

Capítulo 11

ProblemaTipo — un generador de variantes

ProblemaTipo toma una lista de elementos y genera todas las combinaciones posibles, produciendo una variante por cada combinación. Los elementos de la lista pueden ser:

- Cadenas de texto (texto fijo del enunciado).
- Un único Supuesto o Cuestion (aparece igual en todas las variantes).
- Una lista de Supuesto o Cuestion (en cada variante se elige uno de ellos).

```
ProblemaTipo(lista_de_elementos)
```

ProblemaTipo es un iterador: cada llamada a `next()` o cada iteración en un bucle `for` devuelve una tupla (etiqueta, enunciado, cuestiones):

Campo	Contenido
etiqueta	Número de variante como cadena ("1", "2", ...).
enunciado	Texto completo del enunciado de esa variante.
cuestiones	Lista de tuplas (texto, correcto, activa, explicacion) por ítem.

Ejemplo mínimo:

```
from qbank import *

p = ProblemaTipo([
    "Dado que ",
    [
        Supuesto("$\\mathcal{A}$ es verdadero, ", v("A")),
        Supuesto("$\\mathcal{B}$ es verdadero, ", v("B")),
    ],
    "indique qué opción es correcta: ",
    [
        Cuestion("$\\mathcal{A}$ es verdadero.", v("A")),
        Cuestion("$\\mathcal{B}$ es verdadero.", v("B")),
    ],
])

for etiqueta, enunciado, cuestiones in p:
    print(f"Variante {etiqueta}: \n {enunciado}")
    for texto, correcto, activa, exp in cuestiones:
        marca = "✓" if correcto else "×"
        print(f" {texto} {marca} ")
```

Variante 1:

Dado que \mathcal{A} es verdadero, indique qué opción es correcta:
 \mathcal{A} es verdadero. ✓

Variante 2:

Dado que \mathcal{A} es verdadero, indique qué opción es correcta:
 \mathcal{B} es verdadero. ×

Variante 3:

Dado que \mathcal{B} es verdadero, indique qué opción es correcta:

\mathcal{A} es verdadero. ×

Variante 4:

Dado que \mathcal{B} es verdadero, indique qué opción es correcta:

\mathcal{B} es verdadero. ✓

Este ejemplo produce 4 variantes (2 supuestos × 2 cuestiones).

Capítulo 12

ProblemaVF — preguntas de verdadero/falso

ProblemaVF sirve para un formato distinto: un banco de afirmaciones del que se extraen aleatoriamente un subconjunto en cada variante. No usa cálculo proposicional; la corrección de cada ítem se indica directamente con True o False.

```
ProblemaVF(enunciado, cuestiones, NumPreguntas)
```

Parámetro	Descripción
enunciado	Texto del enunciado común a todas las variantes.
cuestiones	Lista de tuplas (texto, correcto).
NumPreguntas	Cuántas afirmaciones se eligen aleatoriamente por variante.

```
enunciado = "Indique qué afirmaciones son verdaderas:"
banco = [
    ("Todo alumno que va a clase aprueba", False),
    ("Todo alumno que sabe aprueba", True),
    ("Si suspenden todos es frustrante", True),
    ("Si aprueban todos eres buen profesor", False),
]
import random
random.seed(42) # para generar siempre el mismo ejemplo
GenVar = iter(ProblemaVF(enunciado, banco, 3)) # 3 ítems por variante

for _ in range(3):
    etiqueta, enunc, cuestiones = next(GenVar)
    print(f"\nVariante {etiqueta}:\n {enunc} ")
    for texto, correcto in cuestiones:
        marca = "✓" if correcto else "×"
        print(f" {texto} {marca}")
```

Variante 1:

```
Indique qué afirmaciones son verdaderas:
Todo alumno que va a clase aprueba ×
Si aprueban todos eres buen profesor ×
Todo alumno que sabe aprueba ✓
```

Variante 2:

```
Indique qué afirmaciones son verdaderas:
Todo alumno que sabe aprueba ✓
Todo alumno que va a clase aprueba ×
Si suspenden todos es frustrante ✓
```

Variante 3:

```
Indique qué afirmaciones son verdaderas:
Todo alumno que va a clase aprueba ×
Si suspenden todos es frustrante ✓
Si aprueban todos eres buen profesor ×
```

Capítulo 13

Preguntas paramétricas con setup

El parámetro opcional `setup` de `ProblemaTipo` permite definir problemas cuyos valores numéricos o simbólicos cambian en cada variante. Cuando se proporciona, en cada iteración:

1. Se llama a `setup()` y el dict que devuelve se convierte en el **espacio de nombres** de la variante.
2. Los textos de los slots que contengan marcadores `@variable` reciben la sustitución correspondiente.
3. Las semánticas y precondiciones que sean callable (`lambda ns: ...`) se evalúan con ese diccionario.

El delimitador `@` se eligió para evitar conflictos con el modo matemático de \LaTeX . Para obtener un arroba literal en el texto, escribe `@@`.

13.1. Sin setup: combinatoria pura

El uso habitual de `ProblemaTipo` (sin `setup`) ya se ha visto en la sección anterior. No hace falta ningún código de `setup`; los textos son fijos y la variación viene solo de la elección entre alternativas.

13.2. Con setup y valores aleatorios

El siguiente ejemplo genera preguntas en las que los valores numéricos cambian en cada iteración porque el `setup` usa `random`:

```
import random

def numeros_aleatorios():
    a = random.randint(1, 9)
    b = random.randint(1, 9)
    return {'a': a, 'b': b, 'suma': a + b}

ejercicio_param = ProblemaTipo(
    [
        "Sean $a = @a$ y $b = @b$. ",
        [
            Cuestion("$a + b = @suma$", True),
            Cuestion("$a + b > 10$", lambda ns: ns['a'] + ns['b'] > 10),
        ],
    ],
    setup=numeros_aleatorios,
)
```

En este ejemplo:

- La cadena "Sean \$a = @a\$ y \$b = @b\$. " interpola `@a` y `@b` con los valores de cada variante.
- La semántica `True` de la primera `Cuestion` indica que la afirmación « $a + b = @suma$ » siempre es correcta (el propio texto incluye ya la respuesta correcta).
- La semántica `lambda ns: ns['a'] + ns['b'] > 10` se evalúa con los valores de la variante.

Importante: cuando `setup` devuelve valores distintos en cada llamada (como aquí, con `random`), el iterador no tiene fin. Controla externamente cuántas variantes generas con un `break`, `itertools.islice` o un `range` acotado.

13.3. Con `setup` y valores fijos

Si el `setup` devuelve siempre el mismo diccionario, la combinatoria sigue siendo finita pero los textos usan los parámetros fijos para componer el enunciado. Esto es útil para parametrizar el mismo tipo de pregunta con constantes que se definen una sola vez:

```
def parametros_fijos():
    return {'n': 3, 'k': 2}

ejercicio_fijo = ProblemaTipo(
    [
        "Considere una matriz de  $n \times k$ . ",
        [
            Supuesto("con rango máximo. ", v("MaxRk")),
            Supuesto("con rango deficiente. ", -v("MaxRk")),
        ],
        [
            Cuestion("Tiene  $k$  columnas linealmente independientes.", v("MaxRk")),
        ],
    ],
    setup=parametros_fijos,
)
```

Capítulo 14

Guardar y cargar problemas en formato JSON

qbank puede convertir cualquier ProblemaTipo a JSON y reconstruirlo después. Esto permite guardar el banco de preguntas en un fichero, compartirlo o cargarlo desde el editor visual sin tener que reescribir el código Python.

14.1. De lista de listas a JSON

Dado un ProblemaTipo creado directamente en Python, basta llamar a `problema_to_dict` para obtener el dict JSON, o a `save_problema` para guardarlo en un fichero:

```
from qbank import *

ejercicio = [
    "Considere ",
    [
        Supuesto("$\\mathcal{A}$", v("A")),
        Supuesto("$\\mathcal{B}$", v("B")),
    ],
    [
        Supuesto("y que $\\mathcal{A}\\rightarrow\\mathcal{C}$", v("A") >> v("C")),
        Supuesto("y que $\\mathcal{B}\\rightarrow\\mathcal{D}$", v("B") ** v("D")),
    ],
    "Conteste: ",
    [
        Cuestion("¿Se da $\\mathcal{C}$?", v("C"),
            exp="Solo si $\\mathcal{A}$ y $\\mathcal{A}\\rightarrow\\mathcal{C}$"),
        Cuestion("¿Se da $\\mathcal{D}$?", v("D")),
    ],
]

p = ProblemaTipo(ejercicio)

import json
d = problema_to_dict(p)
print(json.dumps(d, ensure_ascii=False, indent=2))
```

```
{
  "version": "1",
  "tipo": "ProblemaTipo",
  "nombre": "",
  "setup": null,
  "componentes": [
    "Considere ",
    [
      {
        "tipo": "Supuesto",
        "enunciado": "$\\mathcal{A}$",
        "semantica": "v('A')",
        "precond": "True"
      },
    ],
  ],
}
```

```

    {
      "tipo": "Supuesto",
      "enunciado": "$\mathcal{B}$",
      "semantica": "v('B')",
      "precond": "True"
    }
  ],
  [
    {
      "tipo": "Supuesto",
      "enunciado": "y que $\mathcal{A}\rightarrow\mathcal{C}$.",
      "semantica": "v('A') >> v('C')",
      "precond": "True"
    },
    {
      "tipo": "Supuesto",
      "enunciado": "y que $\mathcal{B}\Leftarrow\mathcal{D}$.",
      "semantica": "v('B') ** v('D')",
      "precond": "True"
    }
  ],
  "Conteste: ",
  [
    {
      "tipo": "Cuestion",
      "enunciado": "¿Se da $\mathcal{C}$?",
      "semantica": "v('C')",
      "precond": "True",
      "exp": "Solo si $\mathcal{A}$ y $\mathcal{A}\rightarrow\mathcal{C}$"
    },
    {
      "tipo": "Cuestion",
      "enunciado": "¿Se da $\mathcal{D}$?",
      "semantica": "v('D')",
      "precond": "True",
      "exp": ""
    }
  ]
]
}

```

Para guardar en un fichero:

```
save_problema(p, "ejemplosManual/ejercicio.json")
```

14.2. De JSON a lista de listas

Para recuperar el problema, `load_problema` devuelve un `ProblemaTipo` listo para iterar:

```

p2 = load_problema("ejemplosManual/ejercicio.json")

for etiqueta, enunciado, cuestiones in p2:
    print(f"Variante {etiqueta}: {enunciado}")
    for texto, correcto, activa, exp in cuestiones:
        marca = "✓" if correcto else "×"
        print(f" {marca} {texto}")

```

Variante 1: Considere \mathcal{A} , y que $\mathcal{A}\rightarrow\mathcal{C}$. Conteste:

✓ ¿Se da \mathcal{C} ?

Variante 2: Considere \mathcal{A} , y que $\mathcal{A}\rightarrow\mathcal{C}$. Conteste:

× ¿Se da \mathcal{D} ?

Variante 3: Considere \mathcal{A} , y que $\mathcal{B}\Leftarrow\mathcal{D}$. Conteste:

× ¿Se da \mathcal{C} ?
 Variante 4: Considere \mathcal{A} , y que $\mathcal{B} \rightarrow \mathcal{D}$. Conteste:
 × ¿Se da \mathcal{D} ?
 Variante 5: Considere \mathcal{B} , y que $\mathcal{A} \rightarrow \mathcal{C}$. Conteste:
 × ¿Se da \mathcal{C} ?
 Variante 6: Considere \mathcal{B} , y que $\mathcal{A} \rightarrow \mathcal{C}$. Conteste:
 × ¿Se da \mathcal{D} ?
 Variante 7: Considere \mathcal{B} , y que $\mathcal{B} \rightarrow \mathcal{D}$. Conteste:
 × ¿Se da \mathcal{C} ?
 Variante 8: Considere \mathcal{B} , y que $\mathcal{B} \rightarrow \mathcal{D}$. Conteste:
 ✓ ¿Se da \mathcal{D} ?

El problema cargado se comporta exactamente igual que el original. También es posible cargar directamente desde un dict sin pasar por fichero, usando `problema_from_dict(d)`.

14.3. Con setup paramétrico

Cuando el problema usa `setup`, el código Python del `setup` debe escribirse como una **cadena de texto** dentro del dict JSON (no como una función Python). De este modo el fichero JSON es autosuficiente: al cargarlo, `load_problema` reconstruye el callable a partir de esa cadena y lo ejecuta al comienzo de cada iteración.

El siguiente ejemplo genera una pregunta sobre rango de matrices en la que las dimensiones n (filas) y k (columnas) cambian en cada variante:

```

from qbank import problema_from_dict, save_problema, load_problema

d = {
    "version": "1",
    "tipo": "ProblemaTipo",
    "nombre": "matriz_rango",
    "setup": (
        "import random\n"
        "n = random.randint(3, 5)\n"
        "k = random.randint(2, n - 1)"
    ),
    "componentes": [
        "Considere una matriz  $A$  de  $n$  filas y  $k$  columnas. ",
        [
            {
                "tipo": "Supuesto",
                "enunciado": "Suponga que  $A$  tiene rango máximo ( $\mathrm{rg}(A)=k$ ). ",
                "semantica": "v('MaxRk')",
                "precond": "True"
            },
            {
                "tipo": "Supuesto",
                "enunciado": "Suponga que  $A$  tiene rango deficiente ( $\mathrm{rg}(A)<k$ ). ",
                "semantica": "-v('MaxRk')",
                "precond": "True"
            }
        ],
        "Indique qué afirmación es correcta: ",
        [
            {
                "tipo": "Cuestion",
                "enunciado": "Las  $k$  columnas de  $A$  son linealmente independientes.",
                "semantica": "v('MaxRk')",
                "precond": "True",
                "exp": ""
            },
            {
                "tipo": "Cuestion",
                "enunciado": " $A^{\top} A$  es invertible.",
                "semantica": "v('MaxRk')",
                "precond": "True",
                "exp": ""
            }
        ]
    ]
}

p = problema_from_dict(d)
save_problema(p, "ejemplosManual/matriz_rango.json")

```

La estructura combina 2 supuestos \times 2 cuestiones = 4 variantes. En cada una, el `setup` llama a `random.randint` de nuevo, por lo que n y k toman valores distintos. Los marcadores `@n` y `@k` en los textos reciben esos valores; la semántica de las cuestiones (`v('MaxRk')`) la evalúa el motor proposicional en función del supuesto activo:

```
import random
random.seed(7)

p2 = load_problema("ejemplosManual/matriz_rango.json")
for etiqueta, enunciado, cuestiones in p2:
    print(f"Variante {etiqueta}: {enunciado}")
    for texto, correcto, activa, exp in cuestiones:
        marca = "✓" if correcto else "×"
        print(f" {marca} {texto}")
```

Variante 1: Considere una matriz A de 4 filas y 2 columnas. Suponga que A tiene rango máximo ($\mathrm{rk}(A) = 2$).
 ✓ Las 2 columnas de A son linealmente independientes.

Variante 2: Considere una matriz A de 4 filas y 2 columnas. Suponga que A tiene rango máximo ($\mathrm{rk}(A) = 2$).
 ✓ $A^{\top} A$ es invertible.

Variante 3: Considere una matriz A de 3 filas y 2 columnas. Suponga que A tiene rango deficiente ($\mathrm{rk}(A) < 2$).
 × Las 2 columnas de A son linealmente independientes.

Variante 4: Considere una matriz A de 4 filas y 2 columnas. Suponga que A tiene rango deficiente ($\mathrm{rk}(A) < 2$).
 × $A^{\top} A$ es invertible.

Nótese que la lógica proposicional y los valores numéricos actúan de forma independiente: `v('MaxRk')` no «sabe» cuánto valen n y k ; su valor de verdad lo determina exclusivamente el supuesto elegido en cada variante. El `setup` solo aporta los números que aparecen en el texto mediante la interpolación `@variable`. Una vez guardado el fichero, `load_problema` reconstituye el `setup` desde la cadena de código y el problema se comporta exactamente igual que el original.

14.4. Limitaciones

- Los `setup` definidos como funciones Python (`def` o `lambda`) no se pueden guardar en JSON porque su código fuente no es recuperable. Si necesitas un `setup` paramétrico y también quieres persistirlo, escribe el código del `setup` como cadena de texto y usa `problema_from_dict` (o el editor visual, que gestiona esto automáticamente).
- `ProblemaVF` se puede guardar y cargar sin restricciones (sus datos son valores directos).

Capítulo 15

Exportar la lista de listas a un fichero .py editable

Además de guardar en JSON, es posible exportar cualquier `ProblemaTipo` como código Python puro: exactamente la lista de listas con `Supuesto` y `Cuestion` que se escribiría a mano. El fichero resultante puede abrirse con cualquier editor de texto y ejecutarse directamente con Python.

```
from qbank import *

p = ProblemaTipo([
    "Dado que ",
    [
        Supuesto("$A$ es verdadero, ", v('A')),
        Supuesto("$B$ es verdadero, ", v('B')),
    ],
    "indique qué opción es correcta: ",
    [
        Cuestion("$A$ es verdadero.", v('A')),
        Cuestion("$B$ es verdadero.", v('B')),
    ],
])

from qbank import problema_to_python
print(problema_to_python(p))
```

```
from qbank import Supuesto, Cuestion, ProblemaTipo
from calccprop import *
```

```
ejercicio = [
    'Dado que ',
    [
        Supuesto('$A$ es verdadero, ', v('A')),
        Supuesto('$B$ es verdadero, ', v('B')),
    ],
    'indique qué opción es correcta: ',
    [
        Cuestion('$A$ es verdadero.', v('A')),
        Cuestion('$B$ es verdadero.', v('B')),
    ],
]
```

```
p = ProblemaTipo(ejercicio)
```

Para guardar directamente en un fichero:

```
from qbank import save_problema_py
save_problema_py(p, "./tmp/mi_problema.py")
```

El fichero `mi_problema.py` es código Python válido que puede editarse y ejecutarse. Si el problema tiene `setup`, la función de generación de valores aleatorios aparece como `def _setup()`: en el fichero exportado, con su cuerpo exactamente como fue guardado en el JSON.

Parte IV

Exportar a AMC (L^AT_EX)

Las funciones AMC generan el código \LaTeX compatible con Auto Multiple Choice. Cada función devuelve una cadena de texto con el bloque $\text{\element}\{...\}$ correspondiente.

Capítulo 16

Funciones disponibles

Función	Descripción
<code>AMC</code>	Pregunta estándar. Incluye solo los ítems con <code>activa=1</code> .
<code>AMC_VF</code>	Pregunta VF. Incluye todos los ítems (activos e inactivos).
<code>AMCProfe</code>	Como <code>AMC</code> pero añade un bloque <code>\explain{}</code> con los rechazados.
<code>AMClastCh</code>	Como <code>AMC</code> pero añade «Ninguna de las anteriores» al final.
<code>AMCmc</code>	Como <code>AMC</code> pero en formato multicolumna (requiere <code>ncols</code>).
<code>AMCmcProfe</code>	Como <code>AMCmc</code> con bloque <code>\explain{}</code> .
<code>AMClastChmc</code>	Como <code>AMClastCh</code> en multicolumna.

Capítulo 17

Firma común

Todas las funciones AMC (excepto las `mc`) tienen la misma firma:

```
AMC(nombre, etiqueta, enunciado, cuestiones, opc=["", ""])
```

Las funciones `mc` añaden el parámetro `ncols` (número de columnas):

```
AMCmc(nombre, etiqueta, enunciado, cuestiones, ncols, opc=["", ""])
```

Parámetro	Descripción
<code>nombre</code>	Nombre del grupo de preguntas (se usa como etiqueta \LaTeX).
<code>etiqueta</code>	Identificador único de esta variante dentro del grupo.
<code>enunciado</code>	Texto del enunciado (obtenido de la tupla generada por <code>ProblemaTipo</code>).
<code>cuestiones</code>	Lista de ítems (obtenida de la tupla generada por <code>ProblemaTipo</code>).
<code>ncols</code>	Número de columnas para el entorno <code>multicols</code> .
<code>opc</code>	Lista [<code>instrucciones_extra</code> , <code>opcion_por_defecto</code>]. Raramente necesario.

Capítulo 18

Ejemplo AMC básico

```
from qbank import *

p = ProblemaTipo([
    "Dado que ",
    [
        Supuesto("$\\mathcal{A}$ es verdadero. ", v("A")),
        Supuesto("$\\mathcal{B}$ es verdadero. ", v("B")),
    ],
    "indique qué opción es correcta: ",
    [
        Cuestion("$\\mathcal{A}$ es verdadero", v("A")),
        Cuestion("$\\mathcal{B}$ es verdadero", v("B")),
    ],
])

nombre = "MiCuestionario"
with open("preguntas.tex", "w") as f:
    for etiqueta, enunciado, cuestiones in p:
        f.write(AMC(nombre, etiqueta, enunciado, cuestiones))
```

```
\\element{MiCuestionario}{
  \\begin{questionmult}{MiCuestionario-1}
  Dado que $\\mathcal{A}$ es verdadero. indique qué opción es correcta:
  \\begin{choices}
\\correctchoice{$\\mathcal{A}$ es verdadero}
  \\end{choices}
  \\end{questionmult} }

\\element{MiCuestionario}{
  \\begin{questionmult}{MiCuestionario-2}
  Dado que $\\mathcal{A}$ es verdadero. indique qué opción es correcta:
  \\begin{choices}
\\wrongchoice {$\\mathcal{B}$ es verdadero}
  \\end{choices}
  \\end{questionmult} }

\\element{MiCuestionario}{
  \\begin{questionmult}{MiCuestionario-3}
  Dado que $\\mathcal{B}$ es verdadero. indique qué opción es correcta:
  \\begin{choices}
\\wrongchoice {$\\mathcal{A}$ es verdadero}
  \\end{choices}
  \\end{questionmult} }

\\element{MiCuestionario}{
  \\begin{questionmult}{MiCuestionario-4}
  Dado que $\\mathcal{B}$ es verdadero. indique qué opción es correcta:
  \\begin{choices}
\\correctchoice{$\\mathcal{B}$ es verdadero}
  \\end{choices}
  \\end{questionmult} }
```

Capítulo 19

Ejemplo AMC multicolumna con «Ninguna de las anteriores»

Usando el mismo p del ejemplo anterior, basta sustituir AMC por AMClastChmc y añadir el número de columnas:

```
with open("preguntas.tex", "w") as f:
    for etiqueta, enunciado, cuestiones in p:
        f.write(AMClastChmc("MiCuestionario", etiqueta, enunciado, cuestiones, ncols=2))
```

Los ítems se distribuyen en dos columnas y se añade automáticamente «Ninguna de las anteriores» como última opción.

Capítulo 20

Integración en el documento AMC

El código generado debe incluirse en el preámbulo del documento principal de AMC:

```
\input{preguntas.tex}
```

Consulta la documentación de AMC para el formato completo del documento.

Parte V

Exportar a Moodle (vía L^AT_EX)

Las funciones `Quiz*` generan un fichero `.tex` que, al compilarlo con `LATEX` (usando el paquete `moodle`), produce un fichero `.xml` importable en Moodle.

Capítulo 21

Flujo de trabajo

Python (qbank) → .tex → pdflatex/xelatex → .xml → Moodle

El paquete \LaTeX `moodle` genera el `.xml` durante la compilación. Para instalarlo:

```
tlmgr install moodle
```

O en distribuciones Debian/Ubuntu:

```
sudo apt install texlive-latex-extra
```

Capítulo 22

Funciones disponibles

Función	Descripción
<code>QuizMoodle</code>	Genera <code>.tex</code> con preguntas de opción múltiple estándar.
<code>QuizMoodleLastCh</code>	Igual, pero añade «Las demás opciones son falsas» al final.
<code>QuizMoodleProfe</code>	Versión para facilitar la revisión por parte del profesor.
<code>QuizMoodleConVariables</code>	Permite plantillas Jinja2 para valores numéricos variables.
<code>QuizMoodleLastChConVariables</code>	Igual, con opción final.
<code>QuizMoodleProfeConVariables</code>	Versión profe con plantillas Jinja2.
<code>QuizVFMoodle</code>	Para preguntas Verdadero/Falso generadas con <code>ProblemaVF</code> .
<code>QuizVFMoodleLastCh</code>	Como <code>QuizVFMoodle</code> con opción final (sobra).

Capítulo 23

Firma común

```
QuizMoodle(nombre, directorio, problema, opc=["", ""])
```

Parámetro	Descripción
<code>nombre</code>	Nombre del cuestionario (aparece en Moodle como categoría).
<code>directorio</code>	Ruta donde se guardará el <code>.tex</code> (debe terminar en <code>/</code>).
<code>problema</code>	Un <code>ProblemaTipo</code> iterable, o un diccionario <code>{nombre: iterable}</code> .
<code>opc</code>	<code>[codigo_latex_extra, texto_opcion_por_defecto]</code> . Raramente necesario.

Capítulo 24

Ejemplo Moodle completo

Creamos el directorio donde vamos a guardar el ejemplo:

```
mkdir -p ejemplosManual
```

El siguiente código genera el fichero de L^AT_EX:

```
from qbank import *

p = ProblemaTipo([
    "Considere ",
    [
        Supuesto("$\\mathcal{A}$ ", v("A")),
        Supuesto("$\\mathcal{B}$ ", v("B")),
    ],
    [
        Supuesto("y que $\\mathcal{A}\\rightarrow\\mathcal{C}$.", v("A") >> v("C")),
        Supuesto("y que $\\mathcal{B}\\rightarrow\\mathcal{D}$.", v("B") ** v("D")),
    ],
    "Indique qué opción es correcta: ",
    [
        Cuestion("Entonces $\\mathcal{C}$ es verdadero", v("C")),
        Cuestion("Entonces $\\mathcal{D}$ es verdadero", v("D")),
    ],
])

QuizMoodleLastCh("EjemploMoodle", "ejemplosManual/", p)
```

Esto crea `salida/EjemploMoodle.tex`. Para obtener el `.xml` y una versión `.pdf` del conjunto de preguntas basta ejecutar:

```
cd ejemplosManual
pdflatex EjemploMoodle.tex
```

Se generan [EjemploMoodle-moodle.xml](#) (para importar en Moodle) y [EjemploMoodle.pdf](#) (vista previa del aspecto de las preguntas).

Capítulo 25

Ejemplo con ProblemaVF para Moodle

El siguiente código genera el fichero de L^AT_EX:

```
from qbank import *

enunciado = "Indique qué afirmaciones son verdaderas:"
banco = [
    ("Todo alumno que va a clase aprueba", False),
    ("Todo alumno que sabe aprueba", True),
    ("Si suspenden todos es frustrante", True),
    ("Si aprueban todos eres buen profesor", False),
    ("Las eñes son letras frecuentes en inglés", False),
]

b = [(codchar(texto), correcto) for texto, correcto in banco]
GenVar = iter(ProblemaVF(codchar(enunciado), b, 2)) # 2 items por variante

QuizVFMoodleLastCh("EjemploVF", "ejemplosManual/", GenVar, num=5) # 5 variantes
```

Para obtener el .xml y una versión .pdf del conjunto de preguntas basta ejecutar:

```
cd ejemplosManual
pdflatex EjemploVF.tex
```

Nota: `codchar()` convierte los caracteres con tilde y la ñ a secuencias L^AT_EX (p.ej. á → \'a). Es necesario para Moodle porque el .tex usa codificación OT1.

Parte VI

Preguntas multi-parte (ProblemaMultiParte)

Los formatos AMC y Moodle permiten agrupar varias sub-preguntas de opción múltiple bajo un único enunciado general. Cada sub-pregunta tiene su propio texto introductorio y su propio bloque de alternativas.

En `qbank`, esto se modela con dos clases nuevas:

- `SubPregunta(intro, cuestiones)`: un texto introductorio y una lista de `Cuestion`. A diferencia de `ProblemaTipo`, las cuestiones *no* se filtran por alternativa: se muestran *todas* con su marca correcto/incorrecto.
- `ProblemaMultiParte(componentes, subpreguntas, setup=None)`: itera igual que `ProblemaTipo`, pero el iterador devuelve `(etiqueta, enunciado_comun, [(intro, cuestiones_eval), ...])` en lugar de una única lista de cuestiones.

Capítulo 26

Ejemplo básico

```
from qbank import *

p = ProblemaMultiParte(
    componentes=[
        "Una desviación típica es un indicador ",
        [Supuesto("de dispersión. ", v('Disp')),
         Supuesto("de tendencia central. ", -v('Disp'))],
    ],
    subpreguntas=[
        SubPregunta("(en cuanto a su objetivo)",
            [Cuestion("de tendencia central", -v('Disp')),
             Cuestion("de dispersión", v('Disp'))]),
        SubPregunta("(en cuanto a su sensibilidad)",
            [Cuestion("sensible a valores extremos", v('Disp')),
             Cuestion("no muy sensible a extremos", -v('Disp'))]),
    ]
)

for etiqueta, enunciado, subpreguntas in p:
    print(f"=== Variante {etiqueta} ===")
    print(f"Enunciado: {enunciado}")
    for intro, cuestiones in subpreguntas:
        print(f" {intro}")
        for texto, correcto, _, _ in cuestiones:
            print(f" { '✓' if correcto else '×' } {texto}")
```

=== Variante 1 ===

Enunciado: Una desviación típica es un indicador de dispersión.

(en cuanto a su objetivo)

- × de tendencia central
- ✓ de dispersión

(en cuanto a su sensibilidad)

- ✓ sensible a valores extremos
- × no muy sensible a extremos

=== Variante 2 ===

Enunciado: Una desviación típica es un indicador de tendencia central.

(en cuanto a su objetivo)

- ✓ de tendencia central
- × de dispersión

(en cuanto a su sensibilidad)

- × sensible a valores extremos
- ✓ no muy sensible a extremos

Capítulo 27

Exportar a AMC

```
with open("preguntas_multi.tex", "w") as f:
    for etiqueta, enunciado, subpreguntas in p:
        f.write(AMC_multipart("Estadistica", etiqueta, enunciado, subpreguntas))

with open("preguntas_multi.tex") as f:
    print(f.read())
```

```
\element{Estadistica}{
\begin{questionmult}{Estadistica-1}
Una desviación típica es un indicador de dispersión.

\emph{(en cuanto a su objetivo)}
{\AMCnoCompleteMulti
\begin{choices}
\wrongchoice {de tendencia central}
\correctchoice{de dispersión}
\end{choices}
}

\emph{(en cuanto a su sensibilidad)}
{\AMCnoCompleteMulti
\begin{choices}
\correctchoice{sensible a valores extremos}
\wrongchoice {no muy sensible a extremos}
\end{choices}
}

\end{questionmult} }

\element{Estadistica}{
\begin{questionmult}{Estadistica-2}
Una desviación típica es un indicador de tendencia central.

\emph{(en cuanto a su objetivo)}
{\AMCnoCompleteMulti
\begin{choices}
\correctchoice{de tendencia central}
\wrongchoice {de dispersión}
\end{choices}
}

\emph{(en cuanto a su sensibilidad)}
{\AMCnoCompleteMulti
\begin{choices}
\wrongchoice {sensible a valores extremos}
\correctchoice{no muy sensible a extremos}
```

```
\end{choices}  
}  
\end{questionmult} }
```

Capítulo 28

Exportar a Moodle (cloze)

```
import os
os.makedirs("ejemplosManual", exist_ok=True)
QuizClozeMulti("Estadistica", "ejemplosManual/", p)
```

Esto genera ejemplosManual/Estadistica.tex. Compila con xelatex para obtener el .xml:

```
cd ejemplosManual
xelatex Estadistica.tex
```

El fichero Estadistica.xml puede importarse en Moodle (Banco de preguntas → Importar → Formato Moodle XML). Cada pregunta cloze contiene tantos `\begin{multi}` como sub-preguntas tenga el ProblemaMultiParte.

Capítulo 29

Persistencia JSON

ProblemaMultiParte se guarda y carga igual que ProblemaTipo:

```
from qbank import save_problema, load_problema

save_problema(p, "ejemplosManual/desv_tipica.json")
p2 = load_problema("ejemplosManual/desv_tipica.json")
```

El formato JSON usa {"tipo": "ProblemaMultiParte", ...} con un campo `subpreguntas` que contiene la lista de sub-preguntas (intro + cuestiones).

Capítulo 30

Exportar a código Python editable

`problema_to_python` y `save_problema_py` también soportan `ProblemaMultiParte`. El fichero generado incluye dos variables: la lista de componentes del enunciado común y la lista de sub-preguntas.

```
from qbank import problema_to_python
print(problema_to_python(p, varname="desv_tipica"))
```

```
from qbank import Supuesto, Cuestion, SubPregunta, ProblemaMultiParte
from calccprop import *
```

```
desv_tipica = [
    'Una desviación típica es un indicador ',
    [
        Supuesto('de dispersión. ', v('Disp')),
        Supuesto('de tendencia central. ', -v('Disp')),
    ],
]
```

```
desv_tipica_subs = [
    SubPregunta('(en cuanto a su objetivo)', [
        Cuestion('de tendencia central', -v('Disp')),
        Cuestion('de dispersión', v('Disp')),
    ]),
    SubPregunta('(en cuanto a su sensibilidad)', [
        Cuestion('sensible a valores extremos', v('Disp')),
        Cuestion('no muy sensible a extremos', -v('Disp')),
    ]),
]
```

```
p = ProblemaMultiParte(desv_tipica, desv_tipica_subs)
```

Parte VII

Editor visual en Jupyter (ProblemaTipoEditor)

`ProblemaTipoEditor` es un formulario interactivo para diseñar problemas de tipo `ProblemaTipo` desde un notebook de Jupyter, sin necesidad de escribir la estructura de listas a mano.

Capítulo 31

Instalación

El editor requiere `ipywidgets`. Instálalo junto con `calcprop-qbank`:

```
pip install "calcprop-qbank[jupyter]"
```

En **JupyterLab 4** es necesario que la extensión `@jupyter-widgets/jupyterlab-manager` esté habilitada. Compruébalo con:

```
jupyter labextension list
```

Si la extensión no aparece (síntoma: el widget se muestra como texto `VBox(children...=)`), ejecuta una sola vez desde el terminal:

```
mkdir -p ~/.local/share/jupyter/labextensions/@jupyter-widgets
ln -sfn $(python -c "import jupyterlab_widgets, os; \
print(os.path.dirname(jupyterlab_widgets.__file__))")/labextension \
~/.local/share/jupyter/labextensions/@jupyter-widgets/jupyterlab-manager
```

Reinicia JupyterLab para que el cambio surta efecto. (En Jupyter Notebook clásico no hace falta este paso.)

Capítulo 32

Uso básico

Abre un notebook con el kernel del entorno donde instalaste `calcprop-qbank`:

```
from qbank import ProblemaTipoEditor
```

Para empezar un problema nuevo:

```
editor = ProblemaTipoEditor()
```

Para cargar un problema desde un fichero JSON existente:

```
editor = ProblemaTipoEditor('mi_problema.json')
```

Capítulo 33

Descripción del formulario

El editor tiene tres zonas:

- **Cabecera:** campo `Nombre` (identificador del problema) y campo `Setup` (código Python del setup paramétrico; déjalo vacío si no necesitas valores variables).
- **Slots:** cada slot es un «componente» del problema. Pulsa + `slot` para añadir slots. Cada slot puede ser de tipo:
 - **texto:** texto fijo que aparece igual en todas las variantes.
 - **alternativas:** una o más filas, cada una con tipo (`Supuesto` / `Cuestion`), enunciado, semántica, precondition y, para `Cuestion`, un campo de explicación/feedback. Pulsa + `alt` para añadir filas.
- **Barra inferior:** controles de acción.

Botón	Acción
<code>Preview</code>	Genera hasta N variantes y las muestra en el área de salida.
<code>Guardar</code>	Serializa el estado actual y lo guarda en el fichero indicado (campo de texto JSON).
<code>Cargar</code>	Lee el fichero indicado y rellena el editor con su contenido.
<code>\{ \}</code> JSON	Muestra el JSON equivalente al estado actual del editor en el área de salida.

Capítulo 34

Ejemplo sin setup

Construir desde el editor el mismo problema del ejemplo de ProblemaTipo:

1. Ejecutar `ProblemaTipoEditor()` en una celda.
2. Dejar el campo `Setup` vacío.
3. + slot → tipo **texto** → escribir "Dado que ".
4. + slot → tipo **alternativas** → + alt dos veces:
 - fila 1: tipo **Supuesto**, enunciado \mathcal{A} es verdadero, =, semántica $v('A')$
 - fila 2: tipo **Supuesto**, enunciado \mathcal{B} es verdadero, =, semántica $v('B')$
5. + slot → tipo **texto** → escribir indique qué opción es correcta: ".
6. + slot → tipo **alternativas** → + alt dos veces:
 - fila 1: tipo **Cuestion**, enunciado \mathcal{A} es verdadero, semántica $v('A')$
 - fila 2: tipo **Cuestion**, enunciado \mathcal{B} es verdadero, semántica $v('B')$
7. Pulsar `Preview` (4 variantes esperadas).
8. Escribir `mi_problema.json` en el campo de ruta y pulsar `Guardar`.

Una vez guardado, el fichero puede cargarse desde código Python:

```
from qbank import load_problema
p = load_problema('mi_problema.json')
for etiqueta, enunciado, cuestiones in p:
    print(etiqueta, enunciado)
```

Capítulo 35

Ejemplo con setup paramétrico

Para un problema con valores variables, escribe en el campo **Setup** el código que genera los valores por variante:

```
import random
a = random.randint(1, 9)
b = random.randint(1, 9)
suma = a + b
```

Luego, en los textos de los slots, usa **@a**, **@b** y **@suma** donde corresponda. Por ejemplo, un slot de tipo **texto** con el contenido "**Sean \$a = @a\$ y \$b = @b\$.** " mostrará los valores aleatorios de cada variante.

Las semánticas que dependan de los valores del setup deben escribirse como lambdas:

```
lambda ns: ns['a'] + ns['b'] > 10
```

Capítulo 36

Obtener el problema desde el editor

Una vez diseñado el problema, puedes obtener el objeto Python directamente desde otra celda:

```
p = editor.to_problema() # ProblemaTipo listo para iterar  
d = editor.to_dict()    # dict JSON equivalente
```

Con `p` ya puedes llamar a las funciones de exportación (`AMC`, `QuizMoodleLastCh`, etc.) como si hubieras definido el problema a mano.

Capítulo 37

Flujo completo: editor → JSON → AMC y Moodle

Este ejemplo muestra el recorrido completo desde el editor visual hasta los ficheros de exportación, pasando por el JSON como punto de guardado intermedio.

37.1. Paso 1 — Diseñar y guardar desde el editor

Crea el problema en el editor y guárdalo en disco (botón `Guardar` o desde código):

```
from qbank import ProblemaTipoEditor

editor = ProblemaTipoEditor()
editor # muestra el formulario
```

Una vez rellenado el formulario, guarda el fichero:

```
editor.save("mi_problema.json")
```

O bien, si prefieres guardarlo a mano:

```
from qbank import save_problema
save_problema(editor.to_problema(), "mi_problema.json")
```

37.2. Paso 2 — Cargar el JSON

En cualquier sesión posterior (o en el mismo notebook, en otra celda):

```
from qbank import *

p = load_problema("mi_problema.json")
```

`p` es un `ProblemaTipo` completamente funcional, idéntico al que se habría obtenido definiendo el problema a mano en Python.

37.3. Paso 3a — Exportar a AMC

```
nombre = "MiCuestionario"

with open("preguntas_amc.tex", "w") as f:
    for etiqueta, enunciado, cuestiones in p:
        f.write(AMC(nombre, etiqueta, enunciado, cuestiones))
```

Variantes habituales:

Función	Cuándo usarla
AMC	Pregunta estándar (solo ítems activos).
AMClastCh	Añade «Ninguna de las anteriores» al final.
AMCmc	Ítems en columnas; requiere <code>ncols</code> .
AMClastChmc	Multicolumna con «Ninguna de las anteriores».

El fichero generado se incluye en el documento principal de AMC con `\input{preguntas_amc.tex}`.

37.4. Paso 3b — Exportar a Moodle

```
QuizMoodleLastCh("MiCuestionario", "salida/", p)
```

Esto genera `salida/MiCuestionario.tex`. A continuación, compila con \LaTeX para obtener el `.xml` importable en Moodle:

```
cd salida
xelatex MiCuestionario.tex
```

El paquete \LaTeX `moodle` produce `MiCuestionario.xml` durante la compilación. Ese fichero es el que se importa en Moodle (Banco de preguntas → Importar → Formato Moodle XML).

Variantes habituales:

Función	Cuándo usarla
QuizMoodle	Pregunta estándar.
QuizMoodleLastCh	Añade «Las demás opciones son falsas» al final.
QuizMoodleProfe	Versión con ítems rechazados visibles (revisión).
QuizVFMoodle	Para ProblemaVF (preguntas de verdadero/falso).

37.5. Paso 4 — Generar múltiples instancias numéricas (problemas con `setup`)

Cuando el problema tiene `setup` con valores aleatorios, iterar una sola vez el problema genera tantas variantes como combinaciones estructurales existan (por ejemplo, 2 supuestos \times 2 cuestiones = 4). Cada variante llama a `setup()` de forma independiente, por lo que cada una puede tener distintos valores numéricos.

Hay dos patrones de exportación masiva:

Opción A — variantes totalmente independientes

Cada una de las 40 variantes tiene sus propios valores numéricos. Útil cuando el banco de preguntas debe ser lo más variado posible.

```
from qbank import *

p = load_problema("mi_problema.json")

with open("preguntas.tex", "w") as f:
    for instancia in range(10):
        for etiqueta, enunciado, cuestiones in p:
            f.write(AMC("MiCuestionario", f"{instancia+1}-{etiqueta}",
                       enunciado, cuestiones))
```

Las 4 variantes estructurales de cada vuelta del bucle externo no comparten necesariamente los mismos valores numéricos entre sí.

Opción B — lotes: las variantes estructurales comparten los mismos valores

Dentro de cada lote, las 4 variantes estructurales usan los mismos valores numéricos. El `setup` se reemplaza temporalmente por un callable que devuelve un dict fijo calculado al inicio del lote.

```
from qbank import *
import random

p = load_problema("mi_problema.json")
d = problema_to_dict(p) # dict reutilizable
```

```

with open("preguntas.tex", "w") as f:
    for instancia in range(10):
        n = random.randint(3, 9)
        k = random.randint(2, n - 1)
        ns = {'n': n, 'k': k}

        p_lote = problema_from_dict(d)
        p_lote.setup = lambda ns=ns: ns # fija n,k para todo el lote

    for etiqueta, enunciado, cuestiones in p_lote:
        f.write(AMC("MiCuestionario", f"{instancia+1}-{etiqueta}",
                    enunciado, cuestiones))

```

El `lambda ns=ns: ns` captura el valor del dict en el momento de creación mediante argumento por defecto, evitando que todas las lambdas apunten al mismo último valor de la variable de bucle.

Opción	Patrón	Valores por variante
A	<code>for i in range(10): for v in p:</code>	Cada variante tiene los suyos
B	<code>p_lote.setup = lambda ns=ns: ns</code>	Las 4 del lote comparten (n, k)

Ambos patrones funcionan igual para Moodle: sustituye `AMC` por `QuizMoodleLastCh` y adapta el nombre del fichero de salida.

Parte VIII

Referencia rápida de la API

Capítulo 38

Clases

```
Supuesto(enunciado, semantica, precond=True)
Cuestion(enunciado, semantica, precond=True, exp="")
ProblemaTipo(lista, setup=None)
    → itera (etiqueta, enunciado, cuestiones)
ProblemaTipoProfe(lista, setup=None)
    → como ProblemaTipo, sin descartar ítems rechazados
ProblemaVF(enunciado, lista, n)
    → itera (etiqueta, enunciado, cuestiones)
SubPregunta(intro, cuestiones)
    → contenedor de sub-pregunta (intro + lista de Cuestion)
ProblemaMultiParte(componentes, subpreguntas, setup=None)
    → itera (etiqueta, enunciado, [(intro, cuestiones), ...])
ProblemaTipoEditor(source=None)
    → editor visual en Jupyter (requiere ipywidgets; solo ProblemaTipo)
```

Los campos `semantica` y `precond` de `Supuesto` y `Cuestion` admiten, además de fórmulas `calcprop`, callables de la forma `lambda ns: ...` para problemas con `setup` paramétrico. Los textos (enunciados) admiten `@variable` para sustituir valores del namespace del `setup`.

Capítulo 39

Funciones JSON

```
problema_from_dict(d)           → ProblemaTipo, ProblemaVF o ProblemaMultiParte
problema_to_dict(problema)     → dict
load_problema(filepath)       → ProblemaTipo, ProblemaVF o ProblemaMultiParte
save_problema(problema, filepath)
load_banco(filepath)          → lista de problemas
save_banco(problemas, filepath) # acepta lista o dict
problema_to_python(problema, varname="ejercicio") → str (ProblemaTipo y ProblemaMultiParte)
save_problema_py(problema, filepath, varname="ejercicio")
```

Capítulo 40

Funciones AMC

```
AMC          (nombre, etiqueta, enunciado, cuestiones, opc=["", ""])
AMC_VF      (nombre, etiqueta, enunciado, cuestiones, opc=["", "Ninguna..."])
AMCProfe    (nombre, etiqueta, enunciado, cuestiones, opc=["", "Ninguna..."])
AMClastCh   (nombre, etiqueta, enunciado, cuestiones, opc=["", "Ninguna..."])
AMCmc       (nombre, etiqueta, enunciado, cuestiones, ncols, opc=["", "Ninguna..."])
AMCmcProfe  (nombre, etiqueta, enunciado, cuestiones, ncols, opc=["", "Ninguna..."])
AMClastChmc (nombre, etiqueta, enunciado, cuestiones, ncols, opc=["", "Ninguna..."])
AMC_multipart(nombre, etiqueta, enunciado, subpreguntas, opc=[""])
    # para ProblemaMultiParte; subpreguntas = [(intro, cuestiones), ...]
```

Capítulo 41

Funciones Moodle

```
QuizMoodle          (nombre, directorio, problema, opc=["", ""])
QuizMoodleLastCh   (nombre, directorio, problema, opc=["", "Las demás..."])
QuizMoodleProfe    (nombre, directorio, problema, opc=["", ""])
QuizVFMoodle       (nombre, directorio, GenVar, num, opc=["", ""])
QuizVFMoodleLastCh (nombre, directorio, GenVar, num, opc=["", "Las demás..."])
QuizMoodleConVariables (nombre, directorio, problema, environment, Valores, opc)
QuizMoodleLastChConVariables (nombre, directorio, problema, environment, Valores, opc)
QuizMoodleProfeConVariables (nombre, directorio, problema, environment, Valores, opc)
QuizClozeMulti     (nombre, directorio, problema, opc=["", ""])
    # para ProblemaMultiParte; genera preguntas tipo cloze
```

Capítulo 42

Operadores de `calccprop` (disponibles tras `from qbank import *`)

```
v("X")           variable proposicional X
~v("X")          negación de X
v("X") & v("Y")  conjunción (X Y)
v("X") | v("Y")  disyunción (X Y)
v("X") >> v("Y") implicación (X → Y)
v("X") ** v("Y") bicondicional (X ↔ Y)
alguno(P, Q, R) al menos una de la lista es verdadera
unoDe(P, Q, R)  exactamente una de la lista es verdadera
True           tautología (siempre verdadero)
False          contradicción (siempre falso)
```

```
test(formula, lista_de_formulas) → bool
```

Devuelve True si formula es consecuencia lógica de la lista de hipótesis.

Parte IX

Preguntas frecuentes

Capítulo 43

¿Por qué algunas variantes no se generan?

Si un Supuesto tiene una precondition que no se satisface en una combinación determinada, esa variante completa se descarta. Es el comportamiento esperado: evita combinaciones de hipótesis incoherentes.

Capítulo 44

¿Cómo incluyo texto L^AT_EX en los enunciados?

Los campos `enunciado` de `Supuesto` y `Cuestion` son cadenas Python ordinarias que se insertan tal cual en el `.tex`. Usa cadenas crudas (`r"..."`) para evitar problemas con las barras invertidas:

```
Cuestion(r"$\mathbf{X}^{\top}\mathbf{X}$ es invertible", -v("Mcoli"))
```

Capítulo 45

¿Para qué sirve `codchar`?

La función `codchar(s)` convierte vocales acentuadas y la ñ a secuencias \LaTeX :

```
codchar("á é í ó ú ñ") # + "\'a \'{e} \'{i} \'{o} \'{u} \{-n}"
```

Es necesaria solo para las funciones Moodle, que generan un `.tex` con codificación OT1. Para AMC no suele hacer falta si el documento principal usa `inputenc` con UTF-8.

Capítulo 46

¿Cómo genero un diccionario de preguntas de varias secciones?

Pasa un diccionario a las funciones Quiz*:

```
problema = {  
    "Sección-A": ProblemaTipo(lista_A),  
    "Sección-B": ProblemaTipo(lista_B),  
}  
QuizMoodleLastCh("MiExamen", "salida/", problema)
```

Cada clave del diccionario se convierte en una categoría dentro del cuestionario Moodle.