

# Índice general

I	Introducción	2
1.	Instalación rápida (para quien ya conoce Python)	4
II	Instalación paso a paso	5
2.	Requisitos previos	6
3.	Qué es un entorno virtual y por qué usarlo	7
4.	Crear y activar un entorno virtual	8
5.	Instalar <code>calcprop</code>	9
6.	Nota para usuarios de Windows	10
III	Variables proposicionales	11
IV	Conectivos lógicos	13
V	La función <code>test</code>	15
7.	Ejemplos básicos	17
8.	<code>test</code> con fórmulas compuestas como hipótesis	18
9.	<code>True</code> y <code>False</code> como fórmulas	19
VI	Funciones auxiliares	20
10.	<code>unoDe</code> — exactamente uno verdadero	21
11.	<code>alguno</code> — al menos uno verdadero	22
VII	Referencia rápida	23

Parte I

**Introducción**

`calcprop` es una librería Python que implementa un motor de cálculo proposicional. Permite construir fórmulas lógicas combinando variables proposicionales con los conectivos habituales, y comprobar si una fórmula es consecuencia lógica de un conjunto de hipótesis.

Está diseñada para usarse como motor de razonamiento en aplicaciones que necesitan determinar automáticamente la veracidad de afirmaciones a partir de supuestos dados. El paquete [calcprop-qbank](#) lo utiliza para generar bancos de preguntas de opción múltiple.

# Capítulo 1

## Instalación rápida (para quien ya conoce Python)

---

```
pip install calcprop
```

---

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

## Parte II

# Instalación paso a paso

## Capítulo 2

# Requisitos previos

calccprop 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](https://python.org) e instálalo (en Windows marca la opción *Add Python to PATH*).

## Capítulo 3

# 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.

## Capítulo 4

# Crear y activar un entorno virtual

Abre una terminal y sitúate en el directorio donde quieres trabajar:

```
mkdir mi_proyecto  
cd mi_proyecto
```

Crea el entorno virtual:

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>

## Capítulo 5

# Instalar calcprop

Con el entorno activo:

---

```
pip install calcprop
```

---

Verifica la instalación:

---

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

---

## Capítulo 6

# Nota para usuarios de Windows

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

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

## Parte III

# Variables proposicionales

Las variables proposicionales son los átomos del cálculo proposicional. Se crean con la función `v()`, que acepta cualquier cadena de texto como nombre:

---

```
from calccprop import *
```

```
A = v("A")  
B = v("B")
```

---

Cada llamada a `v()` con el mismo argumento produce la misma variable:

---

```
print(v("A") == v("A")) # True  
print(v("A") == v("B")) # False
```

---

True  
False

Parte IV

Conectivos lógicos

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

Operador Python	Conectiva	Significado
<code>-v('A')</code>	Negación ( $\neg A$ )	Es falso que A
<code>v('A') &amp; v('B')</code>	Conjunción (A B)	A y B
<code>v('A')   v('B')</code>	Disyunción (A B)	A o B (o ambos)
<code>v('A') &gt;&gt; v('B')</code>	Implicación ( $A \rightarrow B$ )	Si A entonces B
<code>v('A') ** v('B')</code>	Bicondicional (A B)	A si y solo si B

Las fórmulas se pueden anidar libremente:

---

```
f1 = v("A") >> v("B")           # A implica B
f2 = -v("A") | v("B")           # no-A o B (equivalente a f1)
f3 = (v("A") & v("B")) ** v("C") # (A y B) si y solo si C
print(f1)
print(f2)
print(f3)
```

---

```
v('A') >> v('B')
-v('A') | v('B')
v('A') & v('B') ** v('C')
```

## Parte V

# La función test

`test(formula, hipotesis)` comprueba si una fórmula es consecuencia lógica de una lista de hipótesis. Devuelve `True` o `False`.

---

```
test(formula, hipotesis=[])
```

---

Parámetro	Descripción
-----------	-------------

<code>formula</code>	La fórmula cuya derivabilidad se comprueba. También acepta <code>True</code> (tautología) o <code>False</code> (contradicción).
----------------------	---

<code>hipotesis</code>	Lista de fórmulas que se asumen verdaderas. Por defecto vacía.
------------------------	--

# Capítulo 7

## Ejemplos básicos

---

```
# Modus ponens: de A y (A + B) se deriva B
print(test(v("B"), [v("A"), v("A") >> v("B")])) # True

# Sin hipótesis, B no es derivable
print(test(v("B"), [])) # False

# Una tautología siempre es verdadera
print(test(v("A") | ~v("A"), [])) # True

# Una contradicción nunca es derivable
print(test(False, [])) # False
```

---

True  
False  
True  
False

## Capítulo 8

# test con fórmulas compuestas como hipótesis

Las hipótesis pueden ser cualquier fórmula, no solo variables atómicas:

---

```
# De (A → B) y (B → C) se deriva (A → C)
print(test(v("A") >> v("C"),
          [v("A") >> v("B"), v("B") >> v("C")])) # True

# De (A → B) se deriva tanto (A → B) como (B → A)
print(test(v("A") >> v("B"), [v("A") ** v("B")])) # True
print(test(v("B") >> v("A"), [v("A") ** v("B")])) # True
```

---

True

True

True

## Capítulo 9

# True y False como fórmulas

Los literales Python `True` y `False` se pueden usar directamente como fórmulas:

---

```
print(test(True, []))      # True - tautología
print(test(False, []))    # False - contradicción
print(test(v("A"), [True])) # False - True como hipótesis no aporta información
print(test(v("A"), [False])) # True - de una hipótesis falsa se deriva cualquier cosa
```

---

`True`  
`False`

**Parte VI**

**Funciones auxiliares**

# Capítulo 10

## unoDe — exactamente uno verdadero

`unoDe(A, B, C, ...)` construye una fórmula que es verdadera cuando exactamente uno de los argumentos es verdadero:

---

```
# Exactamente uno de A, B, C es verdadero
f = unoDe(v("A"), v("B"), v("C"))

# Si A es verdadero, B y C deben ser falsos
print(test(f, [v("A"), -v("B"), -v("C")])) # True
print(test(f, [v("A"), v("B")])) # False (dos verdaderos)
```

---

True  
False

# Capítulo 11

## alguno — al menos uno verdadero

`alguno(A, B, C, ...)` es equivalente a la disyunción  $A \mid B \mid C \mid \dots$ :

---

```
f = alguno(v("A"), v("B"), v("C"))
print(test(f, [v("B")]))      # True
print(test(f, []))           # False
```

---

True  
False

Parte VII

Referencia rápida

```

from calccprop import *

v("X")          → variable proposicional X
¬v("X")         → negación de X (¬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)
True / False    → tautología / contradicción

test(formula, [h1, h2, ...]) → bool
    True si formula es consecuencia lógica de las hipótesis h1, h2, ...

unoDe(A, B, C, ...) → exactamente uno de los argumentos es verdadero
alguno(A, B, C, ...) → al menos uno de los argumentos es verdadero

```