

King of Dumbs

el rey de los tontos

Nivel: ♦ Intermedio

¿Por qué el ordenador dice que $0,1 + 0,2$ no es $0,3$?

$0,1$ no cabe en binario y el ordenador lo redondea. Qué es la coma flotante, por qué el error se acumula y qué usan la IA y el dinero. En tres niveles y con widget.

Español Publicado 15 de octubre de 2026 Actualizado 16 de octubre de 2026



Escanea para abrir la lección interactiva

<https://kingofdumbs.com/es/lecciones/coma-flotante>

Qué vas a aprender

- Por qué 0,1 no se puede guardar exacto en binario
- La coma flotante: notación científica en binario con cifras fijas
- Cómo se acumula el redondeo y cómo se compara con tolerancia
- float32, float16 y bfloat16 en la IA, y decimales para el dinero

Antes de esta lección

Si algo te suena a chino, estas lecciones lo explican desde cero:

1. [¿Cuánto es \$7 \times 10^{27}\$? Números enormes sin miedo](#)

Símbolos que aparecen

Δ incremento: cuánto ha cambiado (♠) · E_a subíndice: la etiqueta de abajo (♠) · Σ sumatorio: súmalos todos (♠) · x_i índice i: el elemento número i (♠) · $\approx$ aproximadamente igual (♠) · $\leq \geq$ menor o igual / mayor o igual (♠) · $|x|$ valor absoluto: sin signo (♠) · 10^{-3} exponente negativo: dividir (♦♠) · $\log$ logaritmo: ¿cuántas veces multiplico? (♠) · $\lfloor x \rfloor$ parte entera: redondear hacia abajo (♠) · ε épsilon: algo muy pequeño (♠) · δ delta minúscula: un error pequeño (♠) · $\in$ pertenece a (♠)

Abre el navegador en el ordenador, pulsa F12 para abrir la consola, escribe `0.1 + 0.2` y pulsa Enter. La respuesta es `0.30000000000000004`. Prueba en Python: lo mismo. Una máquina hecha para calcular falla una suma de primaria.

La pregunta tonta de esta lección es: **¿por qué el ordenador dice que $0,1 + 0,2$ no es $0,3$?** No está roto. Guarda los números de una manera en la que 0,1 no cabe exacto. Esta lección explica cuál es esa manera, por qué casi nunca se nota y cuándo sí.

Elige tu carta: ♣ si empiezas de cero, ♦ si no te asusta una fórmula sencilla, ♠ si quieres la notación. Puedes cambiarla en cualquier momento.

0,1 no cabe en binario

cada casilla vale

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$
0	0	0	1	1	0	0	1	1

nueve casillas: $\frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} = 0,099609375$

$0,1 = 0,0001100110011\dots_2$ (el 0011 se repite)

El ordenador guarda los números en **binario**, solo con ceros y unos. En decimal, cada cifra después de la coma vale la décima parte que la anterior: décimas, centésimas... En binario, cada cifra vale **la mitad** que la anterior: $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$...

Para escribir 0,1 en binario hay que ver qué trozos caben: $\frac{1}{2}$ no, $\frac{1}{4}$ no, $\frac{1}{8}$ no, $\frac{1}{16}$ sí... Sale 0,0001100110011..., y **el grupo 0011 se repite para siempre**. Con nueve cifras llegas a 0,099609375; con más te acercas, pero nunca terminas. **Le pasa lo mismo que a $\frac{1}{3}$ en decimal**.

Un ordenador normal guarda **53 cifras binarias** y **redondea al número más cercano** que puede escribir con ellas. Su 0,1 vale exactamente 0,1000000000000000055511151231257827021181583404541015625, un poco más que 0,1 (tutorial de Python).

Con 0,2 y 0,3 pasa lo mismo, cada uno con su pequeño error. **La suma** de los dos primeros no cae en el número que el ordenador usa para 0,3, sino en **el siguiente**: 0,30000000000000004.

El Rey de los Tontos dice:

El ordenador suma bien lo que tiene. El problema es que lo que tiene no es 0,1: no le cabe.

Coma flotante: notación científica en binario

signo

exponente (11)

mantisa (52)



+ 2⁻⁴

1,6

0,1 = + 1,6 × 2⁻⁴ · unas 16 cifras decimales

Un número en **coma flotante** se guarda en tres partes, como en notación científica pero en base 2:

- el **signo**, + o -;
- el **exponente**, que dice por qué potencia de 2 se multiplica;
- la **mantisa**, que son las cifras.

El formato normal se llama **binary64** (o “doble precisión”): 64 bits, 1 de signo, 11 de exponente y 52 de mantisa (MDN; NumPy). La mantisa empieza siempre por un 1 que no hace falta guardar, así que tiene 53 cifras binarias, unas 16 decimales. Es el formato de los números de JavaScript, de los float de Python y de Excel. Por ejemplo, $0,1 = + 1,6 \times 2^{-4}$.

Como hay siempre las mismas cifras, **la separación entre dos números seguidos** crece con el tamaño. **Justo por encima de 1 es de $2,2 \times 10^{-16}$** . Junto a 10^{16} es de 2, y por eso $10^{16} + 1$ da 10^{16} . Por lo mismo, los números enteros solo son exactos hasta unos 9×10^{15} (MDN).

Algunas combinaciones de bits se reservan para el cero, **el infinito y los resultados sin sentido**, como la raíz cuadrada de -1 (Microsoft, sobre Excel).

Cuando el resultado de una operación cae entre dos números que se pueden guardar, **se redondea al más cercano**. Cada redondeo se equivoca, como mucho, en media separación: menos de una parte en 9×10^{15} .

El redondeo se acumula

0,1 sumado n veces

1	0,1	6	0,6
2	0,2	7	0,7
3	0,300000000000000004	8	0,7999999999999999
4	0,4	9	0,8999999999999999
5	0,5	10	0,9999999999999999

$10^{16} + 1 = 10^{16}$: el 1 no cabe

Cada operación redondea, y los errores se van acumulando. Al sumar 0,1 en un bucle, en JavaScript:

Veces	Resultado
3	0,300000000000000004
8	0,7999999999999999
10	0,9999999999999999
10 000	1000,0000000001588

Además, **el orden cambia el resultado**. Sumar un número pequeño a uno muy grande puede no cambiar nada ($10^{16} + 1 = 10^{16}$), así que $(a + b) + c$ no siempre da lo mismo que $a + (b + c)$ (Goldberg, 1991).

Tres remedios habituales:

- **Comparar con tolerancia**, en vez de con `==`. En Python, `math.isclose(0.1 + 0.1 + 0.1, 0.3)` da True (tutorial de Python).
- **Sumar con cuidado**. `math.fsum` lleva la cuenta de las cifras que se pierden y **redondea una sola vez**. Y desde Python 3.12, `sum()` **compensa los errores por dentro**: `sum([0.1] * 10) == 1.0` da True en casi todas las instalaciones, mientras que el bucle a mano da 0,9999999999999999.
- **Redondear al mostrar**, no al calcular: dos decimales para un precio.

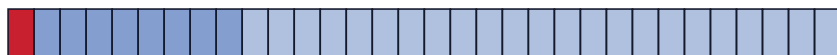
El Rey de los Tontos dice:

Nunca preguntes a un número con decimales si es igual a otro. Pregúntale si está muy cerca.

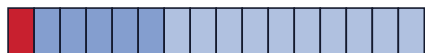


Menos bits para la IA, decimales para el dinero

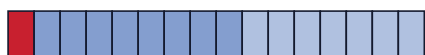
float32 ≈ 7 cifras · hasta $3,4 \times 10^{38}$



float16 ≈ 3 cifras · hasta 65 504



bfloat16 ≈ 2 cifras · hasta $3,4 \times 10^{38}$



Además de binary64 hay formatos más pequeños:

- **float32**, 32 bits (1 + 8 + 23): **unas 7 cifras**.
- **float16**, 16 bits (1 + 5 + 10): unas 3 cifras, y **nada por encima de 65 504**.
- **bfloat16**, 16 bits (1 + 8 + 7): solo unas 2 cifras, pero **el mismo rango que float32**. Lo pensaron en Google para el aprendizaje automático (blog de Google Cloud, 2019).

Las redes neuronales usan mucho estos formatos pequeños: ocupan la mitad de memoria y pueden ir más rápido. El truco se llama **precisión mixta**: hacer las multiplicaciones en 16 bits y guardar una copia de los pesos de la red en 32. En float16, **un cambio pequeño puede perderse** si el número al que se suma es **2048 veces mayor** o más (Micikevicius et al., 2018). Es la báscula de camiones otra vez.

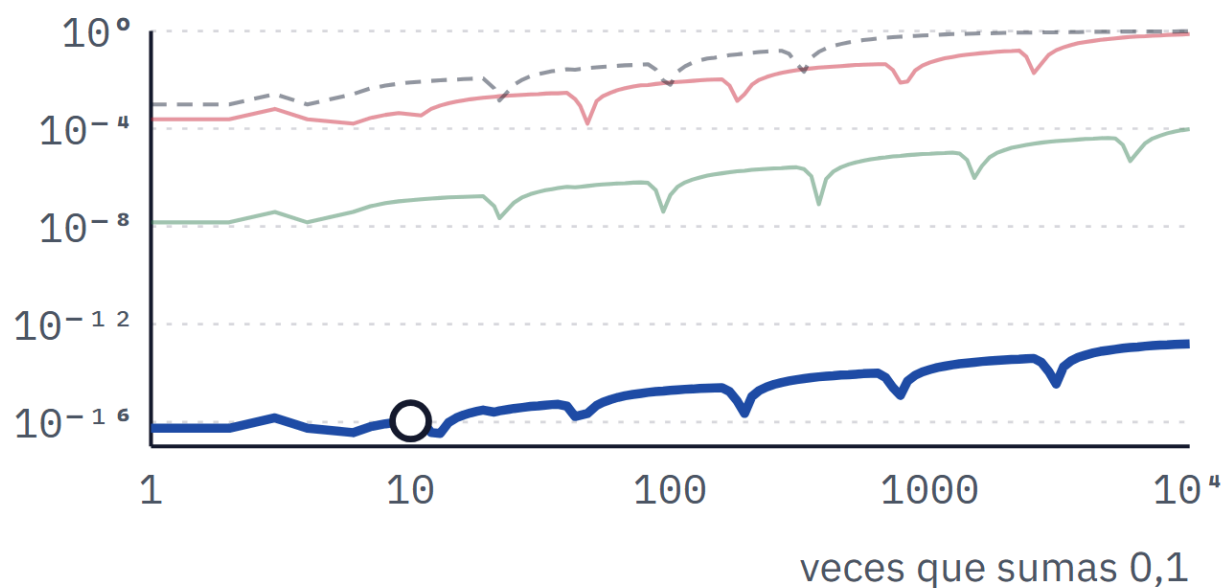
Para el dinero se hace lo contrario: **aritmética decimal**, que cuenta en décimas y centésimas como nosotros, o enteros de céntimos. En decimal, $0,1 + 0,1 + 0,1 - 0,3$ da exactamente 0; en binario, $5,55 \times 10^{-17}$ (documentación del módulo decimal de Python, pensado para contabilidad).



Prueba tú: suma 0,1 una y otra vez

Elige cómo guarda los números el ordenador y cuántas veces suma 0,1. Compara lo que da con lo que debería dar y mira cómo crece el error en la gráfica. Prueba float16 con 5000 veces.

error relativo



Una curva por formato; la del elegido, en grueso.

Cómo guarda los números

- ☒ binary64, lo normal (64 bits)
- ☐ float32 (32 bits)
- ☐ float16 (16 bits)
- ☐ bfloat16 (16 bits, el de la IA)

Veces que sumas 0,1 10 veces

Debería dar 1

Da **0,9999999999999999**

Error $1,1 \times 10^{-16}$

Así se guarda 0,1 0,1000000000000000055511151231257827021181583404541015625

Se desvía $1,1 \times 10^{-16}$: una parte entre 9×10^{15} .



Widget interactivo. Pruébalo en la web: <https://kingofdumbs.com/es/lecciones/coma-flotante>



Reto final

1. ¿Por qué el ordenador no puede guardar 0,1 exacto?

- a. Porque es un número demasiado pequeño
- b. Porque en binario 0,1 tiene infinitas cifras, como 1/3 en decimal
- c. Porque los programas están mal hechos

2. Sumas 0,1 diez veces en un bucle y compruebas si el resultado es igual a 1. ¿Qué pasa?

- a. Sí, es igual a 1
- b. No: da 0,9999999999999999, así que hay que comparar con tolerancia
- c. El programa da un error

3. Para guardar el saldo de una cuenta bancaria, ¿qué es mejor?

- a. float16, para ahorrar memoria
- b. Contar céntimos con números enteros, o usar aritmética decimal
- c. binary64, que nunca redondea



En tres frases

1. El ordenador cuenta en binario, y en binario 0,1 no tiene final, como 1/3 en decimal: se guarda redondeado, un pelín por encima.
2. La coma flotante es notación científica en binario con cifras fijas (unas 16 en binary64): cada operación redondea, el error se acumula y el orden de las sumas importa.
3. Por eso los números con decimales se comparan con tolerancia; la IA usa formatos de 16 bits porque ahorran memoria, y el dinero va en céntimos enteros o en decimal.



Fuentes

- Por qué 0,1 no es exacto, su valor guardado, `math.isclose` y `math.fsum`: Python 3.15, [tutorial, capítulo 15](#) y [módulo math](#). `sum()` con suma de Neumaier: [novedades de Python 3.12](#). Aritmética decimal: [módulo decimal](#).
- Representación, redondeo, no asociatividad y suma de Kahan: D. Goldberg, “What every computer scientist should know about floating-point arithmetic”, *ACM Computing Surveys* 23(1), 5–48 (1991), [doi:10.1145/103162.103163](#), [reimpresión de Oracle](#); W. Kahan, *Communications of the ACM* 8(1), 40 (1965), [doi:10.1145/363707.363723](#). Estándar: [IEEE 754-2019](#).
- Formatos y bits: [MDN, Number](#); [NumPy 2.5, tipos de datos](#); [PyTorch 2.14, torch.dtype](#); [Excel y la coma flotante \(Microsoft\)](#).
- bfloat16: [blog de Google Cloud \(2019\)](#) y [documentación de Cloud TPU](#). Precisión mixta: P. Micikevicius et al., “Mixed Precision Training”, ICLR 2018, [arXiv:1710.03740](#).

Cómo está hecha esta lección: investigada y redactada con ayuda de Claude a partir de las fuentes de arriba, comprobadas y contrastadas con una segunda fuente el 15 de octubre de 2026. Los resultados de sumar 0,1 en un bucle, en cada formato, se midieron en JavaScript (Node 24) y los comprueba un test; las cifras sin enlace propio son cálculos a partir de las fuentes. Los dibujos son propios.

Soluciones

1. **b)** Porque en binario $0,1$ tiene infinitas cifras, como $1/3$ en decimal. Eso es. Cuenta en mitades, y una décima no se acaba nunca.
2. **b)** No: da $0,9999999999999999$, así que hay que comparar con tolerancia. Bien. Cada suma redondea y el error se acumula.
3. **b)** Contar céntimos con números enteros, o usar aritmética decimal. Exacto. Con el dinero no se redondea en binario.

Lección de King of Dumbs, kingofdumbs.com. Versión para leer sin conexión; la original, con widgets que se tocan, está en: <https://kingofdumbs.com/es/lecciones/coma-flotante>

Actualizada el 16 de octubre de 2026.