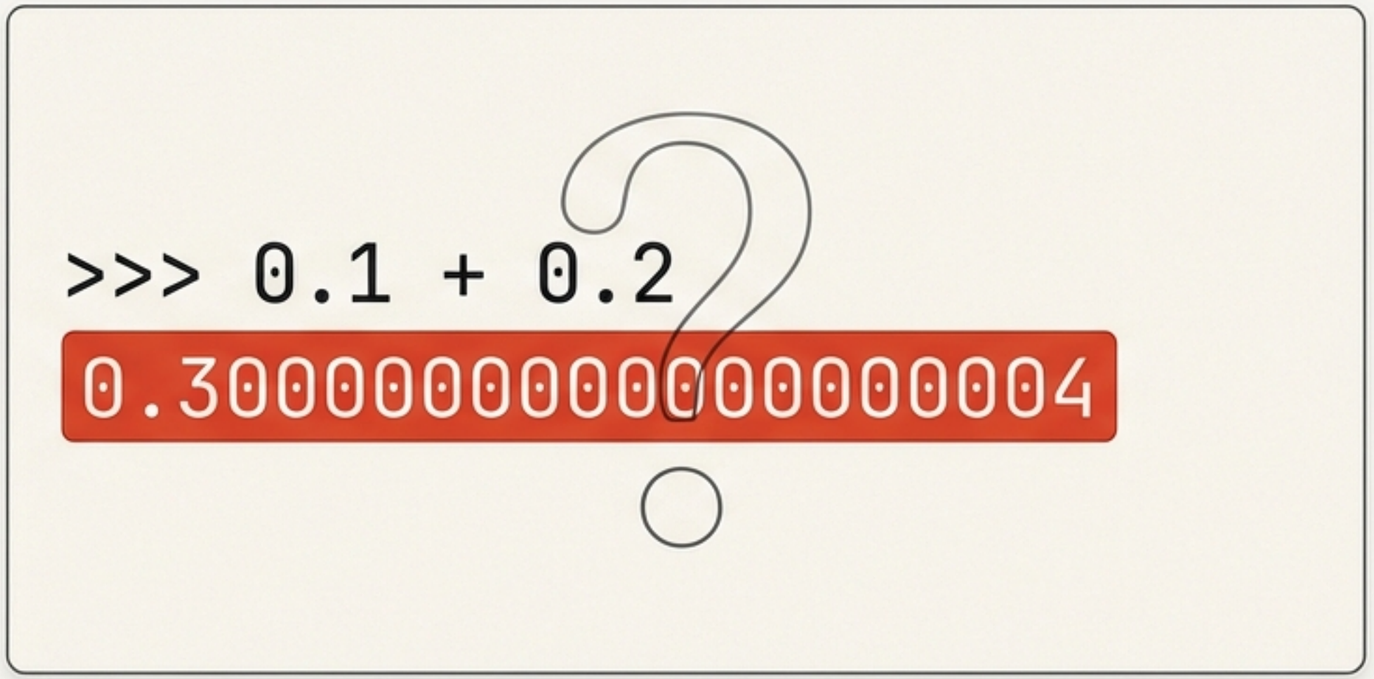
The Hidden Flaw

Why Your Computer Can't Do Simple Math (And Why It Matters)



This isn’t a bug. This is by design. Let’s uncover why.

Our World: The Language of Ten

$$123.45$$

$$= 1\times10^{2} + 2\times10^{1} + 3\times10^{0} + 4\times10^{-1} + 5\times10^{-2}$$

Everything is built from powers of 10.
Simple and intuitive for us.

The Computer's World: The Language of Two

$$101.01_2$$

$$= 1\times2^{2} + 0\times2^{1} + 1\times2^{0} + 0\times2^{-1} + 1\times2^{-2}$$

Everything is built from powers of 2.
All 0s and 1s.

The Flaw in Translation

In Base-10	In Base-2
1/3 = 0.3333333333...	1/10 = 0.0001100110011...

Just as we can’t write 1/3 perfectly,
a computer can’t write 0.1 perfectly.

A Float is a Necessary Approximation

A `float` stores an approximation. It cuts the repeating binary number after a fixed number of bits.

So, `0.1` is actually stored as something incredibly close, but not exact: `0.1000000000000000555...`

The Fundamental Rule of Precision

In Base-10

A fraction is finite only if its denominator's prime factors are **2**s and **5**s.

Finite: 1/2, 1/5, 1/8 (2^3), 1/10 (2×5)

Infinite: 1/3 (Fails because of **3**).

In Base-2

A fraction is finite only if its denominator's prime factors are are **2**s.

Finite: 1/2, 1/4 (2^2)

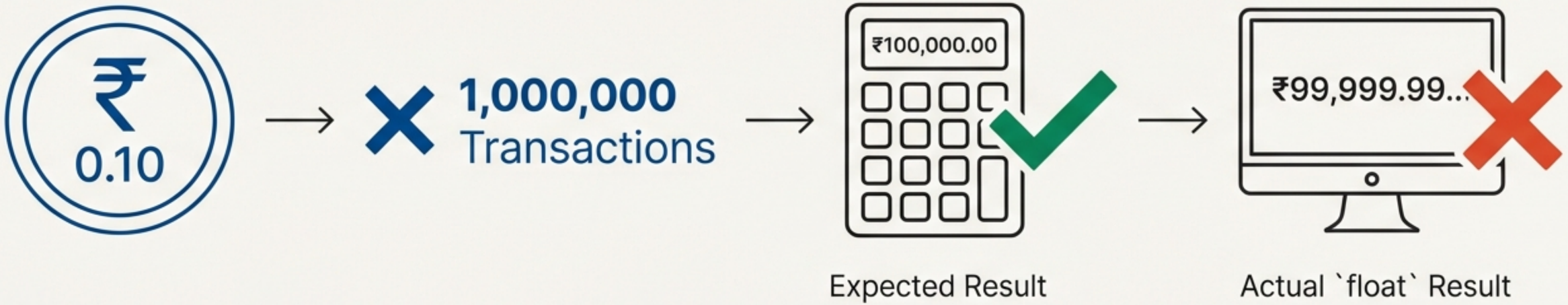
Infinite: 1/10 (which is $1/(2 \times 5)$). It is infinite because of the **5**.

The Crime Scene Revisited

```
   0.1000000000000000555...
+  0.2000000000000001110...
-------------------------------
=  0.3000000000000000004
```

You are not adding `0.1 + 0.2`. You are adding two slightly inaccurate approximations. Their tiny ‘rounding error’ tails combine.

Why This is Unacceptable for Money



Small errors accumulate. Banks don’t tolerate ‘slightly off’.

The Right Tool: Thinking in Base-10 with Decimal

The `Decimal` type performs math the way *we* do—in base-10. It avoids binary translation entirely.

```
from decimal import Decimal
```

- ✓ Numbers are stored and computed in base-10.
- ✓ Decimal fractions stay exact.
- ✓ No infinite binary representation.
- ✓ No truncation. No hidden error.

The Exact Answer

Standard `float`

```
print(0.1 + 0.2)
```

```
0.30000000000000004
```

Python `Decimal`

```
from decimal import Decimal

print(Decimal("0.1") + Decimal("0.2"))
```

```
Decimal('0.3')
```

Why Professional Systems Insist on Precision

AWS
DynamoDB

> DynamoDB stores numbers as exact decimals. Accepting floats would corrupt data. AWS refuses to guess how to round. So they force you to be explicit.

This isn't just a best practice; it's a requirement for data integrity at scale.

Your Simple, Safe Mental Model

‘float’

Fast

Approximate

Unsafe for money & data integrity

‘Decimal’

Slower

Exact

Safe for money & data integrity

Binary cannot represent most decimals exactly. Decimal arithmetic can.

Make It Real

Run this once and never forget it.

```
# The float comparison fails
print(0.1 + 0.2 == 0.3)  False

# The Decimal comparison succeeds
from decimal import Decimal
print(Decimal("0.1") + Decimal("0.2") == Decimal("0.3")) True
```

This is not trivia. This is foundational.