

King of Dumbs

笨蛋之王

级别: ♠ 新手

为什么电脑说 $0.1 + 0.2$ 不等于 0.3 ?

0.1 在二进制里写不完，电脑只好把它舍入。什么是浮点数，误差为什么会累积，AI 和记账又该用什么数。分三个层次讲，配一个互动小工具。

中文 发布于 2026年10月15日 更新于 2026年10月16日



扫码打开可互动的课程

<https://kingofdumbs.com/zh/lessons/floating-point>

你将学到

- 为什么 0.1 无法在二进制中精确存储
- 浮点数：位数固定的二进制科学记数法
- 舍入误差如何累积，以及如何带容差比较
- AI 里的 float32、float16 和 bfloat16，以及记账用的十进制

学这一课之前

如果有什么看不懂，这些课会从零讲起：

1. [7×10²⁷ 到底有多大？轻松搞懂超大数字](#)

本课会出现的符号

Δ 变化量：变了多少 (♠) · E_a 下标：右下角的标签 (♠) · Σ 求和：全部加起来 (♠) · x_i 下标 i：第 i 个元素 (♠) · $\approx$ 约等于 (♠) · $\leq \geq$ 小于等于 / 大于等于 (♠) · $|x|$ 绝对值：去掉正负号 (♠) · 10^{-3} 负指数：表示除 (♦♠) · $\log$ 对数：要乘几次？ (♠) · $\lfloor x \rfloor$ 向下取整 (♠) · ϵ epsilon：很小的量 (♠) · δ 小写 delta：小误差 (♠) · $\in$ 属于 (♠)

在电脑上打开浏览器，按 F12 打开控制台，输入 $0.1 + 0.2$ ，回车。答案是 0.30000000000000004 。换成 Python 试试：一样。一台专门用来计算的机器，竟然算错了一道小学加法。

这一课的傻问题是：**为什么电脑说 $0.1 + 0.2$ 不等于 0.3 ？** 它没坏。它存数字的方式决定了 0.1 存不准。这一课讲清楚这种方式是什么，为什么几乎察觉不到，以及什么时候会露馅。

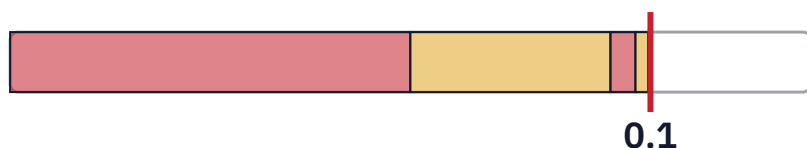
选一张牌：零基础选 ♣，不怕简单公式选 ♦，想看记号选 ♠。随时可以换。

0.1 在二进制里放不下

写不完的数

十进制里： $1 \div 3 = 0.333333\dots$

用“一半的一半”来拼，0.1 也拼不完：



差一点点.....但总是缺一小块

试着用小数写出三分之一： $0.3333\dots$ 不管写多少位，都到不了头，总差一点。这个数本身没什么奇怪的，只是十进制按十分之一、百分之一、千分之一来数，三分之一在这种数法里没有尽头。

电脑不按十分之一来数，它按**一半**来数：一半、四分之一、八分之一、十六分之一.....每一块都是前一块的一半。用这样的小块去拼，十分之一同样拼不完：可以非常接近，但总是缺一小块。

电脑只有几十块的位置，所以拼到那里就停下，留下最接近的那个数。它的 0.1 不是 0.1，而是多了一丁点。它的 0.2 也多了一丁点，而它的 0.3 却少了一丁点。加在一起，这些“一丁点”对不上，结果就成了 0.300000000000000004 。

笨蛋之王说：

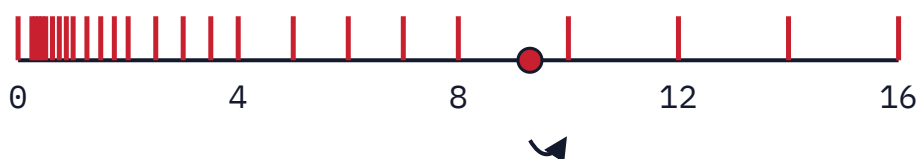
电脑做加法没算错。问题是它手里的数根本不是 0.1：放不下。

浮点数：二进制的科学记数法

刻度很少的尺子

靠近 0，刻度密

离得远，刻度疏



落在两道刻度之间的数，归到最近的那一道

在“很大的数”那一课里，你见过科学记数法： 7×10^{27} 就是 7 后面跟 27 个零。用它可以只写几位数字，就表示很大或很小的数。

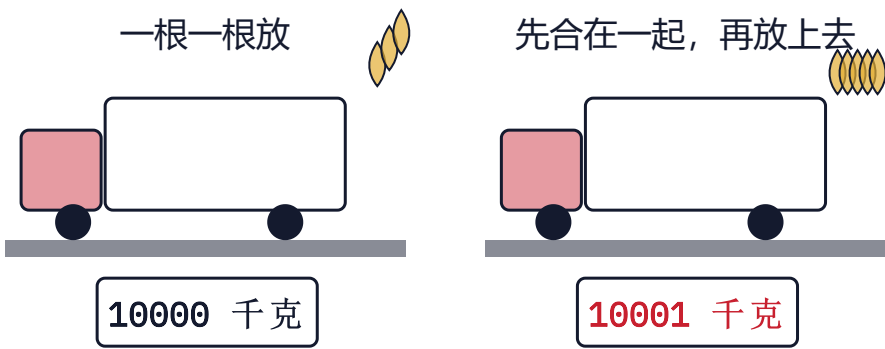
电脑也是这么做的，只不过用二进制，而且位数总是固定的：大约 16 位十进制有效数字。这叫**浮点数**，因为小数点是“浮动”的：它随数的大小移动，而数字的位数始终不变。

想象一把刻度数量固定的尺子。靠近 0 的地方刻度很密，可以把小东西量得很细；离 0 越远，刻度越稀。要量的东西落在两道刻度之间，就记下最近的那一道：这就是**舍入**。

有 16 位数字，这把尺子已经非常精细：0.1 的偏差不到 10^{16} 分之一。所以几乎察觉不到。



舍入误差会累积



同样的羽毛，顺序不同，结果就不同

想象一台有点特别的地磅：它不会把所有东西重新称一遍，而是把新放上去的重量加到已经显示的数字上，再舍入到千克。上面停着一辆 10000 千克的卡车，另外有一千根羽毛，合起来正好 1 千克。放一根羽毛：地磅还是 10000。再放一根：10000。一千根一根一根地放：还是 10000，因为每次都舍入到千克，一根羽毛不够。现在把一千根羽毛装进袋子，一次放上去：10001。

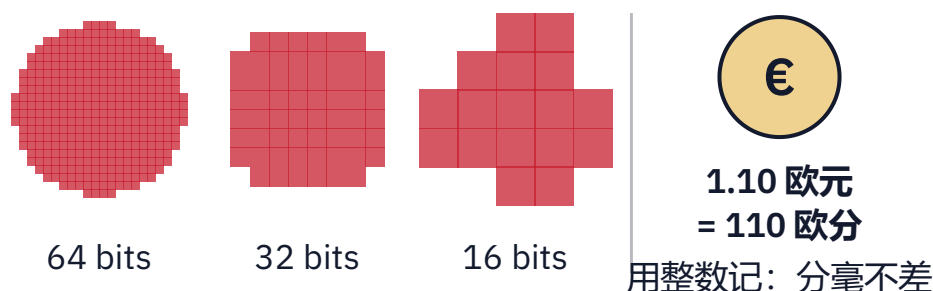
电脑也是这样。每次加法都要舍入，小误差会按照相加的**顺序**累积起来，或者丢掉。把 0.1 连加十次，得到的不是 1，而是 0.9999999999999999。

大多数时候这无所谓，因为误差极小。但如果你拿结果和 1 比较来做决定，答案就是“不相等”。所以，对带小数的数，程序员不问“相等吗？”，而问“足够接近吗？”。

笨蛋之王说：
永远别问一个带小数的数是否等于另一个数。要问它们是否足够接近。



AI 用更少的位，记账用十进制



不是每个数都需要 16 位数字。神经网络，比如机器翻译背后的那种，要用几百万个数做极多的乘法。如果每个数只占一半的空间，内存就能多装一倍，计算也可能更快。代价是每个数更粗糙，就像像素更少的照片。

只有 16 位时，电脑只有大约 3 位有效数字。把 0.1 一遍遍加上去，到 256 时和就卡住了：0.1 还不到尺子上两道刻度间隔的一半，一舍入就没了。下面的小工具里能看到。

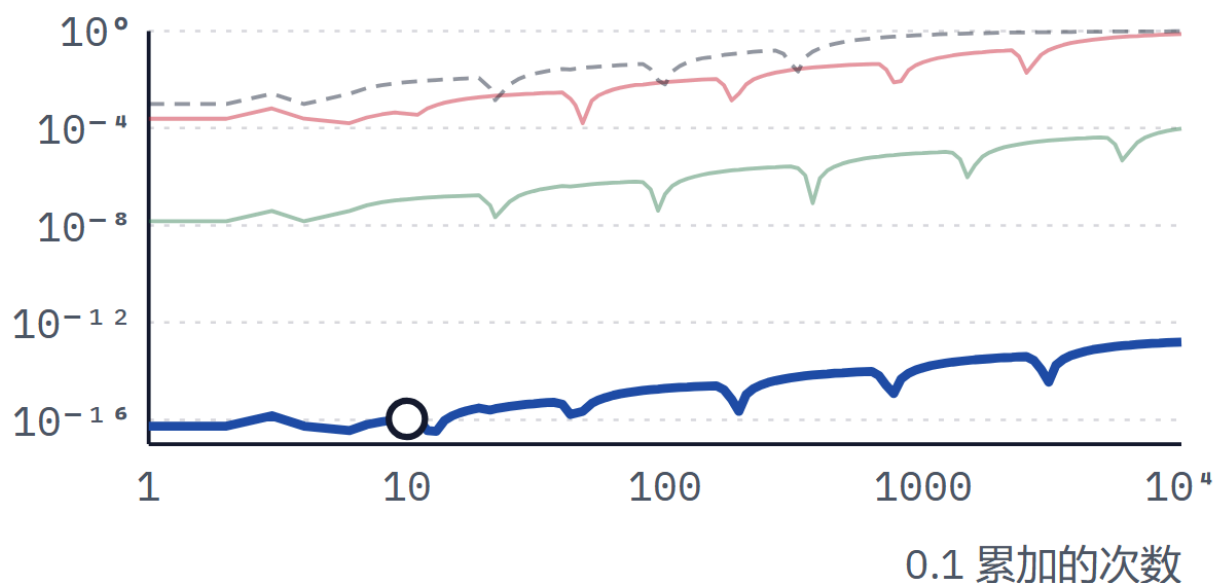
记账则正好相反：一分钱的误差都不行。记账时建议不要用这种二进制小数，而是用整数按“分”来记，整数能精确存储；或者像人一样用十进制来算。



动手试试：把 0.1 加了又加

选择电脑存数字的方式，以及把 0.1 加多少次。比较实际得到的和应该得到的，看看图上的误差怎么增长。试试 float16 加 5000 次。

相对误差



每种格式一条曲线；选中的那条加粗。

数字的存储方式

- ☒ — binary64, 最常用 (64 位)
- ☐ — float32 (32 位)
- ☐ — float16 (16 位)
- ☐ — bfloat16 (16 位, AI 常用)

0.1 累加的次数 10 次

应该得到 1

实际得到 **0.9999999999999999**

误差 1.1×10^{-16}

0.1 实际存成 0.10000000000000000055511151231257827021181583404541015625

偏差 1.1×10^{-16} : 相当于 9×10^{15} 分之一。



互动小工具。在网站上试一试: <https://kingofdumbs.com/zh/lessons/floating-point>



最后的挑战

1. 为什么电脑无法精确存储 0.1?

- a. 因为这个数太小了
- b. 因为 0.1 在二进制里有无限多位，就像十进制里的 1/3
- c. 因为程序写得不好

2. 你在循环里把 0.1 加十次，然后检查结果是否等于 1。会怎样？

- a. 是的，等于 1
- b. 不等于：结果是 0.9999999999999999，所以要带容差比较
- c. 程序会报错

3. 要存一个银行账户的余额，用什么最好？

- a. float16，省内存
- b. 用整数按分来记，或者用十进制算术
- c. binary64，它从来不舍入



三句话总结

1. 电脑用二进制计数，而 0.1 在二进制里没有尽头，就像十进制里的 1/3：它存下的是舍入后的值，稍微偏大一点。
2. 浮点数就是位数固定的二进制科学记数法（binary64 约 16 位）：每次运算都要舍入，误差会累积，求和的顺序也有影响。
3. 所以带小数的数要带容差比较；AI 用 16 位格式是因为省内存，钱则用整数按分记，或者用十进制。



参考资料

- 0.1 为什么存不准、它存下的值、`math.isclose` 和 `math.fsum`: Python 3.15, [教程第 15 章](#) 和 [math 模块](#)。
`sum()` 使用 Neumaier 求和: [Python 3.12 新特性](#)。十进制算术: [decimal 模块](#)。
- 表示、舍入、不满足结合律和 Kahan 求和: 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](#), [Oracle 重印版](#); W. Kahan, *Communications of the ACM* 8(1), 40 (1965), [doi:10.1145/363707.363723](#)。标准: [IEEE 754-2019](#)。
- 格式和位: [MDN, Number](#); [NumPy 2.5, 数据类型](#); [PyTorch 2.14, torch.dtype](#); [Excel 与浮点数 \(微软\)](#)。
- `bf16`: [谷歌云博客 \(2019\)](#) 和 [Cloud TPU 文档](#)。混合精度: P. Micikevicius et al., “Mixed Precision Training”, ICLR 2018, [arXiv:1710.03740](#)。

这一课是怎么做的: 在 Claude 的帮助下, 根据上面的资料调研和撰写, 并于 2026 年 10 月 15 日用第二个来源核对。各种格式下循环累加 0.1 的结果是在 JavaScript (Node 24) 里实测的, 并有测试检查; 没有单独链接的数字是根据资料计算的。插图是原创的。

答案

1. **b)** 因为 0.1 在二进制里有无限多位，就像十进制里的 $1/3$. 对。电脑按一半一半来数，十分之一永远拼不完。
2. **b)** 不等于：结果是 0.9999999999999999，所以要带容差比较. 很好。每次加法都会舍入，误差会累积。
3. **b)** 用整数按分来记，或者用十进制算术. 没错。和钱有关的，不要用二进制舍入。

King of Dumbs 的课程，kingofdumbs.com。这是离线版；可以动手玩的原版在：
<https://kingofdumbs.com/zh/lessons/floating-point>

更新于 2026年10月16日.