🧮 Floating-Point Precision Errors and the `decimal` Module in Python

🔶 Introduction

In Python (and most programming languages), **floating-point numbers** (`float`) are represented in binary using a fixed number of bits. Because of this, certain decimal numbers **cannot be represented exactly**, which leads to small **rounding errors** in arithmetic operations.

🔷 Example: Floating-Point Precision Error

```python
>>> 0.1 + 0.1 + 0.1 0.30000000000000004 >>> 0.1 + 0.1 + 0.1 - 0.3 5.551115123125783e-17
```

📌 Explanation:

- Although mathematically `0.1 + 0.1 + 0.1 = 0.3` , Python returns `0.30000000000000004` .
- When subtracting `0.3` , the result is not `0.0` , but a very small number close to zero (`5.55e-17`).
- This happens because **0.1 cannot be exactly represented** in binary — it's stored as an approximation.

🔷 Why It Happens

- Computers store floating-point numbers using the **IEEE 754** format.
- In this format, numbers like `0.1` or `0.3` do **not have an exact binary equivalent**, just like `1/3` cannot be exactly represented in decimal.

🔶 Solution: Use the `decimal` Module

Python provides the `decimal` module, which stores numbers **as decimal fractions**, not binary, and offers **arbitrary precision**.

✅ Example Using `Decimal` :

```python
>>> from decimal import Decimal >>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1')
Decimal('0.3') >>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1') - Decimal('0.3')
Decimal('0.0')
```

✅ Explanation:

- `Decimal('0.1')` is treated as **exactly 0.1**, not an approximation.
- The result of addition and subtraction is **mathematically accurate**.

🔷 Caution:

If you do this instead:

```python
>>> Decimal(0.1) Decimal('0.1000000000000000055511...')
```

You're converting a **float to Decimal**, which still carries the float's inaccuracy. Always use **strings** (e.g., `'0.1'`) when initializing `Decimal` values to maintain precision.

✅ When to Use `Decimal`:

- **Financial calculations** (e.g., banking, accounting)
- **High-precision scientific work**
- **When exact values are crucial**

📌 Summary Table:

Operation	`float` Output	`decimal.Decimal` Output
`0.1 + 0.1 + 0.1`	`0.30000000000000004`	`0.3`
`0.1 + 0.1 + 0.1 - 0.3`	`5.551115123125783e-17`	`0.0`

🔶 Conclusion

Floating-point arithmetic is **fast but imprecise** due to binary representation. If your application demands **accuracy**, especially with decimal fractions (like money), use the `decimal` module to avoid subtle and critical bugs.