(Not-so) Precise Float Arithmetic in Python

PyOhio 2023

Syed Muhammad Dawoud
Sheraz Ali

whoami

- Software Engineer Contractor @ Arbisoft, Lahore, Pakistan
- Python & Django

Syed M. Dawoud Sheraz Ali

GitHub: DawoudSheraz

Medium: @dawoud.sheraz

LinkedIn: syed-muhammad-dawoud-sheraz-ali

Context

- Guess the outcome

```
>>> 0.1 + 0.2 + 0.1 == 0.4
```

```
>>> 0.1 + 0.1 + 0.1 == 0.3
```

```
>>> 145.95 - 45.45
```

```
>>> 500.001 - 400.001
```

Expectations?

```
>>> 0.1 + 0.2 + 0.1 == 0.4
```
TRUE

```
>>> 0.1 + 0.1 + 0.1 == 0.3
```
TRUE

```
>>> 145.95 - 45.45
```
100.5

```
>>> 500.001 - 400.001
```
100.0

Actual

```
>>> 0.1 + 0.2 + 0.1 == 0.4
```

```
>>> 0.1 + 0.1 + 0.1 == 0.3
```

```
>>> 145.95 - 45.45
```

```
>>> 500.001 - 400.001
```

```
>>> 0.1 + 0.2 + 0.1 == 0.4
True
```

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
```

```
>>> 145.95 - 45.45
100.49999999999999
```

```
>>> 500.001 - 400.001
100.0
```

What’s Happening → Float Point representation in hardware

Float approximations - Base 10

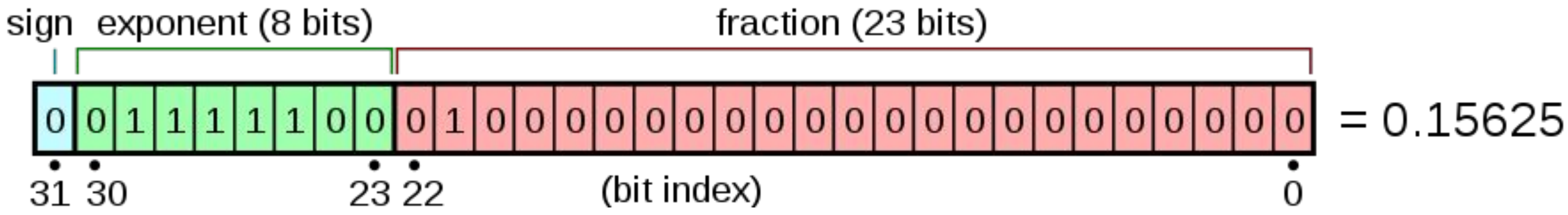
- Combination of finite repeating decimals and infinite repeating decimals
- ½ → 0.50
- ¼ → 0.25
- ⅓ → 0.333 or 0.3333333 or 0.333333333
- ⅙ → 0.166666666667
- 3/11 → 0.2727272727

Float approximations - Base 2

- Like other numbers, floats represented as base-2 on hardware
- Finite and infinite repeating exists in binary as well
- 0.25 → 0.01
- 0.1 → 0.00011001100110011001100...
- 0.1667 → 0.0010101010101100110110011110100000...
- 0.333333 → 0.0101010101010101010100111110111101101011010111...

Float Approximations

- Due to hardware, the approximation issue is part of nearly every programming language
  - IEEE 754 standard
- Rounding off the fractional/mantissa to 23 bits leads to rounding off errors



Workarounds

- Built-in round function
- Takes 2 arguments; number and precision
- Round off manually to avoid unexpected results

```
>>> round(0.1 + 0.1 + 0.1, 2) == round (0.3, 2)
True
```

```
>>> round(145.95-45.45, 1)
100.5
```

Workarounds

- Built-in fractions module
- Represent rational numbers as Fractions
- perform arithmetics on fractions
- Format fractions to floats when needed

```
>>> from fractions import Fraction
>>> fc = Fraction(2, 10)
>>> fc + 1 == Fraction(6, 5)
True
```

```
>>> f"{fc:.2f}"
'0.20'
>>> f"{fc:.4f}"
'0.2000'
```

PyOhio 2023

Workarounds

- Built-in math.isclose
- Verify if the provided values are close to each other
  - Closeness is calculated based on absolute and relative tolerance
- Relative tolerance → maximum allowed difference between values
- Absolute tolerance → minimum absolute difference, useful for comparison near zero

```
>>> isclose(0.1+0.1+0.1, 0.3, rel_tol=0.00000000001)
True
>>> isclose(0.1+0.1+0.2, 0.3, rel_tol=0.00000000001)
False
>>> isclose(0.1+0.1+0.2, 0.3, rel_tol=1.5)
True
```

Unit Testing

- Unittest’s built-in assertions
  - assertAlmostEqual
  - assertNotAlmostEqual
- Signature
  - First, second, decimal places (default 7), msg=None, delta=None
- How does it work?
  - computing the difference of two numbers
  - rounding to the given number of decimal places
  - Comparing to zero
  - Raise assertion depending upon the type
- If delta is supplied, the difference between should be less or equal to or greater than delta.

Closing Thoughts

- Floating approximation is weird and can cause apps to behave unexpectedly
- Align or set expectations explicitly in the code using workarounds to avoid surprises

Links

- Slides → https://github.com/DawoudSheraz/conference-talks/tree/master/pyohio-23
- https://medium.com/python-in-plain-english/mysterious-world-of-pythons-floating-numbers-subtraction-42e157b4bd77
    - My Medium article that inspires this talk
- https://nisal-pubudu.medium.com/how-to-deal-with-floating-point-rounding-error-5f77347a9549
- https://en.wikipedia.org/wiki/IEEE_754
- https://betterprogramming.pub/floating-point-numbers-are-weird-in-python-heres-how-to-fix-them-51336e4ad51a