# Python Numbers: Decimal, Float, Fraction — Q&A (Set 20)

## Q1. Float vs Decimal — pros & cons

Float: IEEE-754 binary64, fast, memory efficient, widely compatible. But many decimal fractions are inexact (0.1), rounding is binary, precision fixed.  
Decimal: base-10 exact decimal fractions, configurable precision/rounding, great for money/accounting. Slower, more memory, fewer libs.

## Q2. Decimal('1.200') vs Decimal('1.2')

Both represent numeric 1.2, but with different internal exponents. Numeric equality, but distinct representations (trailing zeros preserved).

## Q3. Equality check: Decimal('1.200') == Decimal('1.2')

True. Numeric equality. For distinguishing exponents, use compare\_total or compare\_total\_mag.

## Q4. Why start Decimal with string instead of float?

To avoid float rounding error:  
Decimal(0.1) -> Decimal('0.100000000000000005551...')  
Decimal('0.1') -> Decimal('0.1')

## Q5. Mixing Decimal with integers

Works seamlessly: ints convert exactly to Decimal.  
Example: Decimal('2.5') + 3 -> Decimal('5.5')

## Q6. Mixing Decimal with floats

Not allowed directly: raises TypeError. Must convert explicitly using Decimal.from\_float or Decimal(str(...)).

## Q7. Quantity exact with Fraction but not Decimal

1/3 is exact as Fraction(1, 3), not as finite Decimal.

## Q8. Quantity exact with Decimal or Fraction but not float

0.1 is exact with Decimal('0.1') and Fraction(1, 10), but not with binary float.

## Q9. Fraction(1,2) vs Fraction(5,10)

Same internal state. Fraction normalizes to lowest terms, both -> 1/2.

## Q10. Fraction vs int relationship

Containment. Fraction stores two ints (numerator, denominator). It does not inherit from int.