ASSIGNMENT - 20

1. Compare and contrast the float and Decimal classes’ benefits and drawbacks.

Ans: Float vs. Decimal Classes:

* Float: Benefits include compactness, suitable for most general-purpose arithmetic, and wide support in programming languages. However, its drawback lies in limited precision due to how floating-point numbers are stored.
* Decimal: Benefits involve precise decimal arithmetic, avoiding issues with floating-point approximations. Drawbacks include higher memory consumption and slightly slower performance compared to floats.

2. Decimal(‘1.200’) and Decimal(‘1.2’) are two objects to consider. In what sense are these the same object? Are these just two ways of representing the exact same value, or do they correspond to different internal states?

Ans: These are different objects in terms of internal states but represent the same mathematical value. The difference lies in their representation: one preserves the additional trailing zeros while the other represents the value without them.

3. What happens if the equality of Decimal(‘1.200’) and Decimal(‘1.2’) is checked?

Ans: They are considered equal mathematically and would return True if checked for equality (==). The Decimal class handles these representations as equal values.

4. Why is it preferable to start a Decimal object with a string rather than a floating-point value?

Ans: It's preferable to start with a string to avoid precision issues associated with converting floating-point values to Decimal directly, which might introduce inaccuracies due to floating-point approximations.

5. In an arithmetic phrase, how simple is it to combine Decimal objects with integers?

Ans: Combining Decimal objects with integers is straightforward and accurate, providing precise results in arithmetic operations without loss of precision.

6. Can Decimal objects and floating-point values be combined easily?

Ans: Decimal objects and floating-point values can be combined, but care should be taken to convert floating-point values to Decimal objects to maintain precision and avoid floating-point approximation errors.

7. Using the Fraction class but not the Decimal class, give an example of a quantity that can be expressed with absolute precision.

Ans: A quantity like 1/3 can be expressed with absolute precision using the Fraction class as it doesn't suffer from the approximation issues inherent in floating-point representations.

8. Describe a quantity that can be accurately expressed by the Decimal or Fraction classes but not by a floating-point value.

Ans: A repeating decimal like 0.333... or 1/3 can be accurately represented by Decimal or Fraction classes, whereas floating-point values might introduce rounding errors due to their finite precision.

Q9.Consider the following two fraction objects: Fraction(1, 2) and Fraction(1, 2). (5, 10). Is the internal state of these two objects the same? Why do you think that is?

Ans: The internal states of Fraction(1, 2) and Fraction(1, 2) (5, 10) are the same. Fractions are reduced to their simplest form upon instantiation, so both objects represent the same mathematical value despite different initial representations.

Q10. How do the Fraction class and the integer type (int) relate to each other? Containment or inheritance?

Ans: The Fraction class doesn't inherit from the integer type (int) but rather contains integer values. It offers operations that maintain rational numbers without losing precision, unlike integers, which are whole numbers without fractional components.