**ADITYA AMIN**

**ASSIGN :20**

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

the float class is suitable for most general-purpose calculations where performance and memory efficiency are prioritized, but it can introduce rounding errors and limited precision. The Decimal class is more appropriate for applications that require precise decimal calculations, such as financial calculations, with control over precision and rounding, but it comes with a performance overhead and increased memory usage. The choice between the two classes depends on the specific requirements of the task at hand, balancing the need for precision, performance, and interoperability.

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?

In Python, the Decimal('1.200') and Decimal('1.2') are not the same object; they are two distinct Decimal objects. They do not refer to the same memory location in the computer's memory, so they are separate instances.

However, in terms of their values, they represent the same decimal number, which is 1.2. The difference between the two representations lies in their internal states. The Decimal class allows you to specify the number of decimal places explicitly, even if they are trailing zeros.

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

If the equality of Decimal('1.200') and Decimal('1.2') is checked in Python using the == operator, the result will be True. The Decimal class in Python provides a customized implementation of the \_\_eq\_\_ method (equality comparison), which compares the values of two Decimal objects rather than their identities.

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

It is preferable to start a Decimal object with a string rather than a floating-point value in order to avoid potential precision issues and ensure accurate decimal representation.

When a floating-point value is used to initialize a Decimal object, the conversion from the floating-point representation to the decimal representation may introduce rounding errors and loss of precision. This is because floating-point numbers are stored in binary representation, which cannot always represent decimal numbers exactly.

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

In Python, combining Decimal objects with integers in arithmetic operations is straightforward and simple. The Decimal class supports arithmetic operations with integers, allowing you to perform calculations and combine these two types seamlessly.

When you perform arithmetic operations between a Decimal object and an integer, Python automatically handles the conversion and promotes the integer to a Decimal object internally. This allows you to perform arithmetic operations with consistent decimal precision.

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

In Python, combining Decimal objects and floating-point values in arithmetic operations requires some care to ensure accurate results. While it is possible to combine them, you need to be aware of potential precision issues and take precautions to avoid unexpected rounding errors.

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

The Fraction class in Python allows you to represent rational numbers with absolute precision, without any loss of information or rounding errors. By using the Fraction class, you can express quantities precisely in terms of fractions.

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

A quantity that can be accurately expressed by the Decimal or Fraction classes but not by a floating-point value is a repeating decimal or an irrational number.

Let's consider the example of the square root of 2 (√2), which is an irrational number. It cannot be represented exactly as a finite floating-point value due to its infinite and non-repeating decimal expansion. However, both the Decimal and Fraction classes can accurately represent and work with the square root of 2.

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?

The internal state of the two Fraction objects Fraction(1, 2) and Fraction(5, 10) is indeed the same. Both objects represent the fraction 1/2, even though they are created using different numerator and denominator values.

When a Fraction object is created, it automatically reduces the fraction to its simplest form, where the numerator and denominator have no common factors other than 1. This process is known as fraction normalization or fraction simplification.

In the case of Fraction(5, 10), the Fraction class internally simplifies the fraction by dividing both the numerator and denominator by their greatest common divisor (GCD), which is 5 in this case. This simplification results in the normalized fraction Fraction(1, 2).

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

The Fraction class and the integer type (int) in Python do not have a relationship of containment or inheritance. They are distinct types in the Python language.

The Fraction class is part of the fractions module in Python and provides a way to represent and work with rational numbers as fractions. It allows you to create Fraction objects with a numerator and denominator, and perform various arithmetic operations, comparisons, and conversions involving fractions.