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

***Ans***:

1A. Benefits of float:

Float is a built-in Python class and is therefore more lightweight and faster to work with than the Decimal class. Float is more widely supported by Python libraries and frameworks and is the default choice for most numerical calculations in Python. float can represent a wider range of values than Decimal, with a larger exponent range and higher precision for very large or very small numbers.

1B. Drawbacks of float:

Float is not exact and can suffer from rounding errors when performing calculations. This can lead to inaccuracies in calculations and unexpected behaviours in some cases. float uses binary floating-point arithmetic, which can result in some numbers (such as 0.1) being represented imprecisely due to the limited precision of binary floating-point numbers.

2A. Benefits of Decimal:

Decimal is a class from the decimal module in Python and is designed for high-precision decimal arithmetic with exact rounding behaviours. Decimal provides a wider range of precision and rounding options than float, making it well-suited for financial and monetary calculations where exact results are required. Decimal uses decimal floating-point arithmetic, which avoids the imprecisions caused by binary floating-point arithmetic.

2B. Drawbacks of Decimal:

Decimal is slower and more memory-intensive than float, due to the extra overhead required for high-precision decimal arithmetic. Decimal is not as widely supported by Python libraries and frameworks as float and may require additional effort to integrate with existing codebases.

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***:

In Python, Decimal('1.200') and Decimal('1.2') are not the same object, but they do represent the same value. but have slightly different internal states due to their different string representations.

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

***Ans***:

If you check the equality of Decimal('1.200') and Decimal('1.2'), the result would be False.

This is because Decimal('1.200') and Decimal('1.2') have different string representations, even though they represent the same numerical value.

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

***Ans***:

It is generally preferable to create a Decimal object using a string representation rather than a floating-point value because floating-point values can suffer from precision loss and rounding errors due to the way they are represented in binary format.

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

***Ans***:

It is very simple to combine Decimal objects with integers in arithmetic expressions.

When you combine a Decimal object with an integer in an arithmetic expression, the integer is automatically converted to a Decimal object with the same precision and context as the original Decimal object.

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

***Ans***: Decimal objects and floating-point values can be combined in arithmetic expressions, but it requires some care to avoid issues with precision loss and 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.

***Ans***:

The Fraction class in the Python fractions module represents rational numbers as fractions of integers. Since rational numbers can be represented exactly as a ratio of integers, Fraction objects can represent some quantities with absolute precision, even if they cannot represent all decimal values exactly. For example, the fraction 1/3 cannot be represented exactly as a finite decimal, but it can be represented exactly as a Fraction object.

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

***Ans***:

One example of a quantity that can be accurately expressed by the Decimal or Fraction classes but not by a floating-point value is 0.1.

when 0.1 is stored as a floating-point value, it is rounded to the nearest representable value, which is slightly larger than 0.1. This can lead to precision loss and rounding errors when performing arithmetic operations with floating-point values that involve 0.1.

0.1 can be represented exactly as a Decimal or Fraction object, since both classes support arbitrary-precision arithmetic with decimal or rational numbers.

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***:

Yes, the internal state of Fraction(1, 2) and Fraction(5, 10) is the same. This is because both fractions represent the same rational number, which is 1/2.

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

***Ans***:

The Fraction class and the int type in Python are not related by inheritance or containment, as they belong to different class hierarchies. The int type is a built-in numeric type in Python, representing integer values, whereas the Fraction class is defined in the fractions module, representing rational numbers as fractions of integers.