1. Compare and contrast the float and Decimal classes&#39; benefits and drawbacks.

2. Decimal(&#39;1.200&#39;) and Decimal(&#39;1.2&#39;) 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?

3. What happens if the equality of Decimal(&#39;1.200&#39;) and Decimal(&#39;1.2&#39;) is checked?

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

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

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

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

expressed with absolute precision.

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

a floating-point value.

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?

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

inheritance?

### **Q1. Compare and contrast the float and Decimal classes' benefits and drawbacks.**

**Float:**

* **Benefits:**
  + **Performance:** Floating-point operations are generally faster due to hardware support.
  + **Memory Efficiency:** Floats use less memory than Decimal objects.
  + **Wide Support:** Floats are widely used and supported by most numerical libraries and functions.
* **Drawbacks:**
  + **Precision Issues:** Floats are subject to rounding errors because they use binary representation, which can lead to inaccuracies, especially with very small or very large numbers.
  + **Imprecision with Repeating Decimals:** Floats cannot accurately represent some decimal fractions (e.g., 0.1).

**Decimal:**

* **Benefits:**
  + **Precision:** Decimal provides exact decimal representation, making it ideal for financial and other applications where precision is critical.
  + **Customizable Precision:** You can adjust the precision and rounding behavior to suit specific needs.
  + **Exact Arithmetic:** Operations on Decimal objects avoid the precision issues inherent in floating-point arithmetic.
* **Drawbacks:**
  + **Performance:** Decimal operations are slower than floating-point operations due to their software-based implementation.
  + **Memory Usage:** Decimal objects typically consume more memory than floats.
  + **Complexity:** Using Decimal can add complexity to the code, especially when interacting with libraries that expect floats.

### **Q2. 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?**

Decimal('1.200') and Decimal('1.2') represent the same numerical value (1.2), but they correspond to different internal states. The Decimal class preserves the exact number of decimal places specified in the input, so Decimal('1.200') retains the three decimal places, while Decimal('1.2') retains only one. This difference can be significant in contexts where the number of decimal places carries meaning, such as in financial applications.

### **Q3. What happens if the equality of Decimal('1.200') and Decimal('1.2') is checked?**

If you check the equality of Decimal('1.200') and Decimal('1.2'), they will be considered equal because the Decimal class compares numerical values rather than their string representations. Despite having different internal representations, the numerical values are identical, so Decimal('1.200') == Decimal('1.2') will return True.

### **Q4. 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 because floating-point values are inherently imprecise and may introduce rounding errors. When you pass a float to the Decimal constructor, the float's imprecise representation can carry over into the Decimal object, defeating the purpose of using Decimal for precision. Starting with a string ensures that the Decimal object represents the value exactly as intended.

### **Q5. In an arithmetic phrase, how simple is it to combine Decimal objects with integers?**

Combining Decimal objects with integers in an arithmetic expression is straightforward and behaves as expected. The integer is automatically converted to a Decimal object, and the operation is carried out with the precision of the Decimal class. For example, Decimal('1.5') + 2 will return Decimal('3.5').

### **Q6. Can Decimal objects and floating-point values be combined easily?**

Combining Decimal objects and floating-point values is not recommended because it can lead to precision issues. While Python allows you to combine them, the float will first be converted to a Decimal, potentially introducing imprecision. For example, Decimal('1.5') + 0.1 may not yield the precise result you expect due to the float's inherent imprecision.

### **Q7. Using the Fraction class but not the Decimal class, give an example of a quantity that can be expressed with absolute precision.**

A quantity like 1/3 can be expressed with absolute precision using the Fraction class but not the Decimal class. The Fraction class represents the number exactly as a ratio of two integers (Fraction(1, 3)), whereas the Decimal class would require a finite number of decimal places to represent it, which would result in an approximation (e.g., Decimal('0.3333...')).

### **Q8. Describe a quantity that can be accurately expressed by the Decimal or Fraction classes but not by a floating-point value.**

A quantity like 0.1 can be accurately expressed by the Decimal class (Decimal('0.1')) or the Fraction class (Fraction(1, 10)), but not by a floating-point value. In binary floating-point representation, 0.1 is an imprecise value, leading to potential rounding errors in calculations.

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

The internal state of Fraction(1, 2) and Fraction(5, 10) is the same. This is because the Fraction class automatically reduces fractions to their simplest form upon creation. Both Fraction(1, 2) and Fraction(5, 10) are reduced to Fraction(1, 2) internally, so they represent the same numerical value and have the same internal state.

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

The Fraction class and the int type relate to each other through containment rather than inheritance. The Fraction class does not inherit from int; instead, it contains two int objects (the numerator and denominator) as its internal representation. This relationship allows Fraction to represent rational numbers as a ratio of two integers, distinct from the integer type itself.