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

Answer:

Both the float and decimal types store numerical values in Python and they are used for floating-point arithmetic.

|  |  |  |
| --- | --- | --- |
|  | Benefits | Drawbacks |
| Float | It is faster and more memory-efficient than Decimal. It can represent a wider range of numbers than Decimal, including very large or very small values. The math module and many other built-in functions and libraries in Python work with float by default. | It has limited precision and can suffer from rounding errors, which can be a problem in some scientific and financial applications.  It cannot represent some numbers exactly, such as 0.1, due to the way that floating-point numbers are stored in binary.  Comparing float values for equality can be tricky due to rounding errors. |
| decimal | It provides much greater precision than float, making it suitable for applications where accuracy is critical. It can represent decimal numbers exactly, avoiding the rounding errors that can occur with float. It provides control over rounding and precision. | It is slower and more memory-intensive than float. It has a more limited range of representable numbers than float. Some libraries and functions in Python do not work with Decimal values by default, requiring extra effort to use them. |

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?

Answer:

The objects Decimal('1.200') and Decimal('1.2') are not the same object in Python, as they correspond to different internal representation or internal states as they have decimal values of different precisions though they are identical value wise.

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

Answer:

The equality of Decimal('1.200') and Decimal('1.2') is True, however the precision is different.

from decimal import \*

print(Decimal('1.200') == Decimal('1.2'))

It returns True.

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

Answer:

The floating-point values are represented as approximations in memory due to their binary nature. This can result in rounding errors and loss of precision when performing arithmetic operations with decimal numbers. On the other hand, when a Decimal object is created from a string, the value is represented exactly, which means there is no loss of precision or rounding errors during arithmetic operations.

from decimal import \*

print(Decimal(0.9))

-> 0.90000000000000002220446049250313080847263336181640625

print(Decimal('0.9'))

-> 0.9

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

Answer:

Python automatically converts integers to Decimal objects when they are used in operations with Decimal objects. We can do it with use of Decimal().

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

Answer:

It is possible to combine Decimal objects and floating-point values while it may not always result in the expected outcome due to the limitations of floating-point precision. When performing arithmetic operations like adding, subtracting or multiplying between Decimal objects and floating-point values, the floating-point values are automatically converted to Decimal objects. However, this conversion may result in rounding errors and loss of precision. Therefore, it is generally recommended to stick to Decimal objects or use string inputs for more precise and consistent arithmetic operations.

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

Answer:

from fractions import Fraction

a = Fraction(1, 9)

print(a)

-> 1/9

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

Answer:

The value of 1/3 can be accurately expressed by the Decimal or Fraction classes but not by a floating-point.

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?

Answer:

The internal state of these two objects are same. This is because when fractions are created, they are automatically simplified to their lowest terms. In this case, both fractions 1/2 and 5/10 are equivalent, and when created with the Fraction class, they are reduced to the same internal state of 1/2.

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

Answer:

The Fraction class and integer type(int) are related in form of a container. The Fraction class and the integer type (int) are two separate classes in Python's class hierarchy. However, the Fraction class can take an integer as an argument and create a new fraction object from it. For example, Fraction(5) will create a fraction object with the value 5/1. Additionally, arithmetic operations can be performed between fractions and integers, with the integer being automatically converted to a fraction object with a denominator of 1.