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

**Decimal class**

* Provides support for fast correctly-rounded decimal floating point arithmetic.
* It gives the provison to represent the number in a representative way so that it is clearly undertood by the user and give you the exact arithmatic operation result like we use to do normally,
* Object exposes all required parts of the standard. When needed, we can change the rounding and signal handling. This includes an option to enforce exact arithmetic by using exceptions to block any inexact operations.
* Objects share many properties with the other built-in numeric types such as float and int. All of the usual math operations and special methods apply.
* It can be constructed from integers, strings, floats, or tuples. Construction from an integer or a float performs an exact conversion of the value of that integer or float.
* It is faster way to get precise value which we human work and have a smaller range than the float
* Decimal types are better suited for financial and monetary calculationsThe Decimal has a default precision of 28 places

**Floating class**

* It do not represent the exact precise value It gives the value close to the result.
* The float object exposes only a modest portion of its capabilities Some values cannot be exactly represented in a float data type.
* For instance, storing the 0.1 value in float (which is a binary floating point value) variable we get only an approximation of the value.
* Similarly, the 1/3 value cannot be represented exactly in decimal floating point type.Float types is best suited for scientific calculationsThe float has default precision of 18 places.

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 print(Decimal('1.200'))

print(Decimal('1.2'))

Though they are identical value wise, internal representation or internal state of these two

values are different, as they have decimal values of different precisions.

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

Ans It returns that the values are stored are True.

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

OUTPUT

True

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

Ans floating-point value is converted to Decimal format. Decimal can store float value with absolute

precision. But when float value is given as Decimal object,it first has to be converted from floating point value which might already have rounding error.Hence it is preferable to start a Decimal object with a string.

e.g.

import decimal

from decimal import Decimal

a=Decimal(0.3) 0.3 is a float

print(a) It is not stored exactly as 0.3 but as printed.

b=Decimal('0.3') '0.3' is a string

print(b) It exactly prints 0.3, precisely, with correct precision.

OUTPUT

0.299999999999999988897769753748434595763683319091796875

0.3

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

Ans import decimal

from decimal import Decimal

val=2

print(type(val))

a=Decimal(val)

print(a)

b=a\*Decimal(val)

print(b)

output

<class 'int'>

2

4

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

Ans Arithmetic operfations like Adding,subtracting or multiplying a Decimal object by a

floating-point value is generates an error. To do these operations, the floating point has to be converted to a Decimal object—for example, converting from a floating-point value and then rounding.

Else, arithmetic operations between the two types cause runtime errors.

import decimal

from decimal import Decimal

val=0.3

b=Decimal(val)

print("Muliplication:",Decimal('0.2')\*b ) Multiplying a decimal object with float value converted to Decimal

print(Decimal('0.2')\*val) Error when Decimal obeject is multiplied by a float

output

Muliplication: 0.05999999999999999777955395075

TypeError: unsupported operand type(s) for \*: 'decimal.Decimal' and 'float'

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

Ans e.g

from fractions import Fraction

val=0.5

fr=Fraction(val)

print(fr)

output

1/2

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

Ans d=Decimal('0.1') \* Decimal('0.1')

print("decimal=",d)

frac=Fraction('1/10') \* Fraction('1/10')

print("fraction=",frac)

fl=0.1\*0.1

print("float value=",fl)

OUTPUT

decimal= 0.01

fraction= 1/100

float value= 0.010000000000000002

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

from fractions import Fraction

frac1=Fraction(1, 2)

print(frac1)

frac2=Fraction(5, 10)

print(frac2)

if (frac1 == frac2 ):

print('Fraction (1,2) and Fraction(5,10) are equal')

The internal state of both are same as Fraction(5,10) is reduced to simplest form.

Hence 1/2 is printed in both the cases.

OUTPUT

1/2

1/2

Fraction (1,2) and Fraction(5,10) are equal

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

Ans Fraction class and integer type(int) are related in form of a container.

It contains two ints, one the numerator and the other the denominator.

from fractions import Fraction

frac = Fraction(1,2)

print('numerator is', frac.numerator,type(frac.numerator))

print('denominator is', frac.denominator,type(frac.numerator) )

OUTPUT

numerator is 1 <class 'int'>

denominator is 2 <class 'int'>