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

* Float is a single precision (32 bit) floating point data type and decimal is a 128-bit floating point data type.
* Floating point data type represent number values with fractional parts.
* Decimal accurately represent any number within the precision of the decimal format, whereas Float cannot accurately represent all numbers.
* Decimal used within financial applications that require a high degree of accuracy and easy to avoid rounding errors whereas Float used when you stores scientific numbers and for better performance.
* Performance of Decimals is slower than and float data types.

Float stores an approximate value and decimal stores an exact value. In summary, exact values like money should use decimal, and approximate values like scientific measurements should use float. When multiplying a non integer and dividing by that same number, decimals lose precision while floats do not.

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?

from decimal import \*

Decimal(1.200)

Decimal('1.1999999999999999555910790149937383830547332763671875')

Decimal(1.2)

Decimal('1.1999999999999999555910790149937383830547332763671875')

type(Decimal(1.200))

decimal.Decimal

type(Decimal(1.2))

decimal.Decimal

id(1.200)

140030189443344

id(1.2)

140030189442640

Decimal('1.200')

Decimal('1.200')

Decimal('1.2')

Decimal('1.2')

id(Decimal('1.200'))

140029764136160

id(Decimal('1.2'))

140029764135152

They are the same object in the sense that they have the same value and same type. But internally they are different and have different memory id location. 1.200 is precise upto three decimal places and 1.2 is just precise upto one decimal place.

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

Decimal(1.200) == Decimal(1.2)

True

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

True

They will come out to be equal.

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

So what happens when we write 0.1 in Python? Let's take a look:

print(f"{0.1:.20f}") # 0.10000000000000000555

For those of you not familiar with the syntax above, the :.20f is a way of telling Python we want 20 digits after the decimal point for this float.

As we can see, we don't actually get 0.1: we get a close approximation of 0.1. Unfortunately, sometimes a close approximation just isn't good enough.

one of the easiest ways to make a Decimal object with fractional components is using a string. We just need to pass a string representation of the number to Decimal and it'll take care of the rest:

x = decimal.Decimal("0.1")

print(x) # 0.1

print(f"{x:.20f}") # 0.10000000000000000000

As we can see, printing x to 20 decimal places here gives us 19 zeroes: we don't end up with some random 5s at the end like we did when using float.

If you need a precise decimal representation of a number, using strings to create your Decimal objects is a very simple way to achieve this.

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

Can mix integers and floats freely in operations. Integers and floating-point numbers can be mixed in arithmetic. Python 3 automatically converts integers to floats as needed.

Decimal objects cannot generally be combined with floats or instances of fractions.Fraction in arithmetic operations: an attempt to add a Decimal to a float, for example, will raise a TypeError. However, it is possible to use Python’s comparison operators to compare a Decimal instance x with another number y. This avoids confusing results when doing equality comparisons between numbers of different types.

Changed in version 3.2: Mixed-type comparisons between Decimal instances and other numeric types are now fully supported.

Python can do decimal calculations, too, approximately. Even a number that is actually an integer can be represented in the float type if a decimal point is included.

Decimal('1.200')+1

Decimal('2.200')

Decimal objects and integers can be easily combined.

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

Decimal objects cannot generally be combined with floats or instances of fractions.Fraction in arithmetic operations: an attempt to add a Decimal to a float, for example, will raise a TypeError. However, it is possible to use Python’s comparison operators to compare a Decimal instance x with another number y. This avoids confusing results when doing equality comparisons between numbers of different types.

Decimal('1.200')+1.5

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.

This module provides support for rational number arithmetic. It allows to create a Fraction instance from integers, floats, numbers, decimals and strings. Fraction Instances : A Fraction instance can be constructed from a pair of integers, from another rational number, or from a string. Fraction instances are hashable, and should be treated as immutable.

class fractions.Fraction(numerator=0, denominator=1) : This requires that numerator and denominator are instances of numbers. Rational and a fraction instance with value = (numerator/denominator) is returned. A zerodivision error is raised if denominator = 0.

from fractions import Fraction

print (Fraction(11, 35))

# returns Fraction(11, 35)

print (Fraction(10, 18))

# returns Fraction(5, 9)

print (Fraction())

# returns Fraction(0, 1)

class fractions.Fraction(other\_fraction) : This requires that other\_fraction is instance of numbers.Rational and a fraction instance with same value is returned.

class fractions.Fraction(float) : This requires the float instance and a fraction instance with same value is returned.

from fractions import Fraction

print (Fraction(1.13))

# returns Fraction(1272266894732165, 1125899906842624)

class fractions.Fraction(decimal) : This requires the decimal instance and a fraction instance with same value is returned.

from fractions import Fraction

print (Fraction('1.13'))

# returns Fraction(113, 100)

class fractions.Fraction(string) : This requires the string or unicode instance and a fraction instance with same value is returned. Form for this instance : [sign] numerator [‘/’ denominator] Here, sign represents ‘+’ or ‘-’ and numerator and denominator are strings of single digits.

from fractions import Fraction

print (Fraction('8/25'))

# returns Fraction(8, 25)

print (Fraction('1.13'))

# returns Fraction(113, 100)

print (Fraction('3/7'))

# returns Fraction(3, 7)

print (Fraction('1.414213 \t\n'))

# returns Fraction(1414213, 1000000)

limit\_denominator(max\_denominator=1000000) :

This method is useful for finding rational approximations to a given floating-point number.

This module finds and returns the closest Fraction to self that has denominator at most max\_denominator.

This module can also be used to return the numerator of a given fraction in the lowest term by using the numerator property and the denominator by using the denominator property.

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

Any number that has infinite number of digits such as 1/3, the square root of 2 and PI cannot be represented completely.

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?

Fraction(1, 2)

Fraction(1, 2)

Fraction(5, 10)

Fraction(1, 2)

Fraction(1, 2) == Fraction(5, 10)

True

id(Fraction(1, 2))

140029764218160

id(Fraction(5, 10))

140030189461904

Internally they are not the same. They have different ids.

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

The Fraction class inherits from the abstract base class numbers.