Resolución del ejercicio 1

Crear una clase `NumeroFraccionario` en algún lenguaje de programación, a elección, que permita manejar fracciones con numerador y denominador, implementando las operaciones básicas de suma, resta, multiplicación y división.

Resolución:

Se creará una clase **`NumeroFraccionario`** en **Python.**

1. **Clase `NumeroFraccionario` en Python. Constructor y atributos:** Se crea la clase `NumeroFraccionario` con un constructor que inicializa los atributos numerador y denominador.

```
class NumeroFraccionario:
   def __init__(self, numerador, denominador):
        if denominador == 0:
          raise ValueError("El denominador 0")
       self.numerador = numerador
       self.denominador = denominador
```

2. ***Getter* y *setter* para numerador:** Se implementan los métodos `getNumerador` y `setNumerador` para obtener y modificar el valor del numerador, respectivamente y respetar el encapsulamiento.

```
class NumeroFraccionario:

   def getNumerador(self):
       return self.numerador

   def setNumerador(self, numerador):
       self.numerador = numerador
```

3. ***Getter* y *setter* para denominador:** Se implementan *getter* y *setter* para el denominador. Se valida que **no pueda ser 0** para asegurar la integridad del objeto.

```
class NumeroFraccionario:

   def getDenominador(self):
       return self.denominador

   def setDenominador(self, denominador):
       if denominador != 0:
           self.denominador = denominador
       else:
           raise ValueError("Denominador 0")
```

4. **Comportamiento auxiliar:** Se necesitará calcular el máximo común divisor para poder simplificar la fracción.

```
class NumeroFraccionario:

   def calcularMCD(self, a, b):
       return math.gcd(a, b)

   def simplificarFraccion(self):
       mcd = self.calcularMCD(self.numerador, self.denominador)
       self.numerador //= mcd
       self.denominador //= mcd
```

5. Se implementa la **suma:**

```
class NumeroFraccionario:

   def sumar(self, otraFraccion):

       nuevoDenominador = self.denominador *
                          otraFraccion.denominador

       nuevoNumerador = (self.numerador * otraFraccion.denominador)
                      + (otraFraccion.numerador * self.denominador)

       return NumeroFraccionario(nuevoNumerador, nuevoDenominador)
```

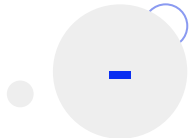
6. Se implementa la **resta:**

```
class NumeroFraccionario:

   def restar(self, otraFraccion):
       nuevoDenominador = self.denominador * otraFraccion.denominador

       nuevoNumerador = (self.numerador * otraFraccion.denominador)
                        - (otraFraccion.numerador * self.denominador)

       return NumeroFraccionario(nuevoNumerador, nuevoDenominador)
```



7. Se implementa la **multiplicación:**

```
class NumeroFraccionario:

   def multiplicar(self, otraFraccion):

       nuevoNumerador = self.numerador * otraFraccion.numerador

       nuevoDenominador = self.denominador * otraFraccion.denominador

       return NumeroFraccionario(nuevoNumerador, nuevoDenominador)
```

8. Se implementa la **división:**

```
class NumeroFraccionario:

   def dividir(self, otraFraccion):

       if otraFraccion.numerador == 0:
           raise ValueError("No se puede dividir por cero.")

       nuevoNumerador = self.numerador * otraFraccion.denominador

       nuevoDenominador = self.denominador * otraFraccion.numerador

       return NumeroFraccionario(nuevoNumerador, nuevoDenominador)
```

9. Se implementa una batería de **pruebas unitarias:**

```
import unittest

class TestNumeroFraccionario(unittest.TestCase):

   def setUp(self):
       self.fraccion1 = NumeroFraccionario(3, 5)
       self.fraccion2 = NumeroFraccionario(2, 7)

   def test_sumar(self):
       resultado = self.fraccion1.sumar(self.fraccion2)
       self.assertEqual(resultado.getNumerador(), 31)
       self.assertEqual(resultado.getDenominador(), 35)

   def test_restar(self):
       resultado = self.fraccion1.restar(self.fraccion2)
       self.assertEqual(resultado.getNumerador(), 11)
       self.assertEqual(resultado.getDenominador(), 35)
```

¡Terminaste el módulo!

Todo listo para rendir el examen