Task 1

from math import gcd  
  
  
class Rational:  
 def \_\_init\_\_(self, numerator=1, denominator=1):  
  
 self.numerator = numerator  
 self.denominator = denominator  
  
 @property  
 def numerator(self):  
 return self.\_\_numerator  
  
 @numerator.setter  
 def numerator(self, value):  
 if not isinstance(value, int):  
 raise TypeError("must be int")  
 self.\_\_numerator = value  
  
 @property  
 def denominator(self):  
 return self.\_\_denominator  
  
 @denominator.setter  
 def denominator(self, value):  
 if not isinstance(value, int):  
 raise TypeError("must be int")  
 if not value:  
 raise ZeroDivisionError()  
 self.\_\_denominator = value  
  
 def \_\_str\_\_(self):  
 return f"({self.\_\_numerator}, {self.\_\_denominator})"  
  
 @staticmethod  
 def reduce\_fraction(n, m):  
 # find greatest common factor of two integers  
  
 k = int(gcd(n, m))  
 return (n // k, m // k)  
  
 def get\_element\_division(self):  
 *"""print cool rational type"""* return f"{self.\_\_numerator}/{self.\_\_denominator}" if self.\_\_numerator else "0"  
  
 def get\_result\_division(self):  
 *"""show result of rational expression"""* return f"{round(self.\_\_numerator / self.\_\_denominator, 3)}"  
  
 @staticmethod  
 def \_\_get\_default\_num\_denum(this, other):  
  
 other\_numerator = 0  
 other\_denominator = 0  
  
 if isinstance(other, Rational):  
  
 other\_numerator = other.\_\_numerator  
 other\_denominator = other.\_\_denominator  
  
 elif isinstance(other, int):  
  
 other\_numerator = other  
 other\_denominator = 1  
 else:  
 raise TypeError("should be Rational type or int")  
  
 return other\_numerator, other\_denominator  
  
 @staticmethod  
 def \_\_add\_template(this, other):  
  
 other\_numerator, other\_denominator = Rational.\_\_get\_default\_num\_denum(  
 this, other  
 )  
  
 denominator = int(  
 this.\_\_denominator  
 \* other\_denominator  
 / gcd(this.\_\_denominator, other\_denominator)  
 )  
 numerator = int(  
 denominator / this.\_\_denominator \* this.\_\_numerator  
 + denominator / other\_denominator \* other\_numerator  
 )  
 return numerator, denominator  
  
 def \_\_add\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_add\_template(self, other)  
  
 result = Rational.reduce\_fraction((numerator), (denominator))  
 return Rational(result[0], result[1])  
  
 def \_\_iadd\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_add\_template(self, other)  
  
 self.\_\_numerator, self.\_\_denominator = Rational.reduce\_fraction(  
 (numerator), (denominator)  
 )  
 return self  
  
 @staticmethod  
 def \_\_sub\_template(this, other):  
  
 other\_numerator, other\_denominator = Rational.\_\_get\_default\_num\_denum(  
 this, other  
 )  
  
 denominator = int(  
 this.\_\_denominator  
 \* other\_denominator  
 / gcd(this.\_\_denominator, other\_denominator)  
 )  
 numerator = int(  
 denominator / this.\_\_denominator \* this.\_\_numerator  
 - denominator / other\_denominator \* other\_numerator  
 )  
 return numerator, denominator  
  
 def \_\_sub\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_sub\_template(self, other)  
 result = Rational.reduce\_fraction((numerator), (denominator))  
 return Rational(result[0], result[1])  
  
 def \_\_isub\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_sub\_template(self, other)  
  
 self.\_\_numerator, self.\_\_denominator = Rational.reduce\_fraction(  
 (numerator), (denominator)  
 )  
 return self  
  
 @staticmethod  
 def \_\_mul\_template(this, other):  
 other\_numerator, other\_denominator = Rational.\_\_get\_default\_num\_denum(  
 this, other  
 )  
  
 denominator = this.\_\_denominator \* other\_denominator  
 numerator = this.\_\_numerator \* other\_numerator  
 return numerator, denominator  
  
 def \_\_mul\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_mul\_template(self, other)  
 result = Rational.reduce\_fraction((numerator), (denominator))  
 return Rational(result[0], result[1])  
  
 def \_\_imul\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_mul\_template(self, other)  
 self.\_\_numerator, self.\_\_denominator = Rational.reduce\_fraction(  
 (numerator), (denominator)  
 )  
 return self  
  
 @staticmethod  
 def \_\_truediv\_template(this, other):  
 other\_numerator, other\_denominator = Rational.\_\_get\_default\_num\_denum(  
 this, other  
 )  
  
 denominator = this.\_\_denominator \* other\_numerator  
 numerator = this.\_\_numerator \* other\_denominator  
 return numerator, denominator  
  
 def \_\_truediv\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_truediv\_template(self, other)  
 result = Rational.reduce\_fraction((numerator), (denominator))  
 return Rational(result[0], result[1])  
  
 def \_\_itruediv\_\_(self, other):  
  
 numerator, denominator = Rational.\_\_truediv\_template(self, other)  
 self.\_\_numerator, self.\_\_denominator = Rational.reduce\_fraction(  
 (numerator), (denominator)  
 )  
 return self  
  
 def \_\_eq\_\_(self, other):  
  
 return not self.\_\_sub\_\_(other).\_\_numerator  
  
  
  
  
a = Rational(3, 9)  
a \*= 2  
print(a, end="\n\n")  
a \*= Rational(3, 9)  
print(a, end="\n\n")

Task 2

from calendar import monthrange  
  
  
class Date:  
 def \_\_init\_\_(self, days, months, years):  
  
 self.\_\_days = days  
 self.\_\_months = months  
 self.\_\_years = years  
  
 @staticmethod  
 def \_\_check\_value(value):  
 if not isinstance(value, int):  
 raise TypeError("Value should be int")  
 if not value > 0:  
 raise ValueError("Must be positive")  
  
 @property  
 def days(self):  
 return self.\_\_days  
  
 @property  
 def months(self):  
 return self.\_\_months  
  
 @property  
 def years(self):  
 return self.\_\_years  
  
 @days.setter  
 def days(self, value):  
 if not Date.\_\_check\_value(value):  
 self.\_\_days = value  
  
 @months.setter  
 def months(self, value):  
 if not Date.\_\_check\_value(value):  
 self.\_\_months = value  
  
 @years.setter  
 def years(self, value):  
 if not Date.\_\_check\_value(value):  
 self.\_\_years = value  
  
  
class Calendar:  
 def \_\_init\_\_(self, day, month, year):  
  
 self.\_\_day = day  
 self.\_\_month = month  
 self.\_\_year = year  
  
 def \_\_str\_\_(self):  
 return f"Calendar({self.day}, {self.month}, {self.year})"  
  
 @staticmethod  
 def \_\_check\_value(value):  
 if not isinstance(value, int):  
 raise TypeError("Value should be int")  
 if not value > 0:  
 raise ValueError("Must be positive")  
  
 @property  
 def day(self):  
 return self.\_\_day  
  
 @property  
 def month(self):  
 return self.\_\_month  
  
 @property  
 def year(self):  
 return self.\_\_year  
  
 @day.setter  
 def day(self, value):  
 if not Calendar.\_\_check\_value(value):  
 self.\_\_day = value  
  
 @month.setter  
 def month(self, value):  
 if not Calendar.\_\_check\_value(value):  
 self.\_\_month = value  
  
 @year.setter  
 def year(self, value):  
 if not Calendar.\_\_check\_value(value):  
 self.\_\_year = value  
  
 @staticmethod  
 def \_\_change\_date\_add(calendar\_object):  
 *"""change add date in case if it wrong"""* if calendar\_object.month > 12:  
 calendar\_object.year += 1  
 calendar\_object.month %= 12  
 if (  
 monthrange(calendar\_object.year, calendar\_object.month)[1]  
 < calendar\_object.day  
 ):  
  
 calendar\_object.month += 1  
 try:  
 calendar\_object.day %= monthrange(  
 calendar\_object.year, calendar\_object.month - 1  
 )[1]  
 except Exception:  
 calendar\_object.month %= 12  
 calendar\_object.year += 1  
 return calendar\_object  
  
 @staticmethod  
 def \_\_change\_date\_sub(calendar\_object):  
 *"""change sub date in case if it wrong"""* if calendar\_object.month <= 0:  
 calendar\_object.year -= 1  
 calendar\_object.month += 12  
 if calendar\_object.day <= 0:  
  
 calendar\_object.month -= 1  
 try:  
 calendar\_object.day += monthrange(  
 calendar\_object.year, calendar\_object.month  
 )[1]  
 except Exception:  
 calendar\_object.month += 12  
 calendar\_object.year -= 1  
 return calendar\_object  
  
 @staticmethod  
 def \_\_get\_added\_date(calendar\_object):  
 *"""get completed added date"""* if (  
 calendar\_object.month > 12  
 or monthrange(calendar\_object.year, calendar\_object.month)[1]  
 < calendar\_object.day  
 ):  
 return Calendar.\_\_get\_added\_date(  
 Calendar.\_\_change\_date\_add(calendar\_object)  
 )  
  
 return calendar\_object  
  
 @staticmethod  
 def \_\_get\_subed\_date(calendar\_object):  
 *"""get completed subed date"""* if calendar\_object.month <= 0 or calendar\_object.day <= 0:  
 return Calendar.\_\_get\_subed\_date(  
 Calendar.\_\_change\_date\_sub(calendar\_object)  
 )  
  
 return calendar\_object  
  
 def \_\_iadd\_\_(self, other):  
 if not isinstance(other, Date):  
 raise TypeError("Must be date type")  
  
 year = self.year + other.years  
 month = self.month + other.months  
 day = self.day + other.days  
 self = Calendar.\_\_get\_added\_date(Calendar(day, month, year))  
 return self  
  
 def \_\_isub\_\_(self, other):  
 if not isinstance(other, Date):  
 raise TypeError("Must be date type")  
  
 if self.\_\_lt\_\_(other):  
 raise ValueError("date should be lower that current date")  
  
 year = self.year - other.years  
 month = self.month - other.months  
 day = self.day - other.days  
  
 self = Calendar.\_\_get\_subed\_date(Calendar(day, month, year))  
 return self  
  
 def \_\_gt\_\_(self, other):  
 if not isinstance(other, Date):  
 raise TypeError("Must be date type")  
  
 return (self.year, self.month, self.day) > (  
 other.years,  
 other.months,  
 other.days,  
 )  
  
 def \_\_ge\_\_(self, other):  
 if not isinstance(other, Date):  
 raise TypeError("Must be date type")  
  
 return (self.year, self.month, self.day) >= (  
 other.years,  
 other.months,  
 other.days,  
 )  
  
 def \_\_lt\_\_(self, other):  
 return not self.\_\_gt\_\_(other)  
  
 def \_\_le\_\_(self, other):  
 return not self.\_\_ge\_\_(other)  
  
 def \_\_eq\_\_(self, other):  
 if not isinstance(other, Date):  
 raise TypeError("Must be date type")  
  
 return (self.year, self.month, self.day) == (  
 other.years,  
 other.months,  
 other.days,  
 )  
  
 def \_\_ne\_\_(self, other):  
 return not self.\_\_eq\_\_(other)  
  
  
a = Calendar(31, 3, 2003)  
b = Date(2, 10, 2000)  
  
print(a >= b)  
print(a <= b)  
print(a == b)  
print(a > b)  
  
a += b  
print(a)  
  
a -= b  
print(a)