Lista 1 - Kacper Budnik

October 19, 2021

https://github.com/KacperBudnik/AiSD/tree/main/List%201

1 Największy wspólny dzielnik

[1]:
```
def Greatest_common_divisor(a,b):
    while a!=b:
        if a>b:
            if b==1:
                a=1
            else:
                if a>b*10**3:
                    p=4
                    while a>b*10**p:
                        p+=1
                    p-=1
                    a-=b*10**p
                else:
                    a-=b
        else:
            if a==1:
                b=1
            else:
                if b>a*10**3:
                    p=4
                    while b>a*10**p:
                        p+=1
                    p-=1
                    b-=a*10**p
                else:
                    b-=a
    return a
```

2 Podstawowa klasa

[2]:

```
class Fraction:
    def __init__(self, Num, Dem):
        if type(Num)!=int or type(Dem)!= int:
            raise TypeError("Muszą być liczby całkowite")

        if Dem==0:
            raise ValueError("Mianownik musi być różny od zera")

        gcd=Greatest_common_divisor(abs(Num),abs(Dem))
        self.sign = 1 if Num*Dem>0 else -1 if Num*Dem<0 else 0
        self.num=abs(Num)//gcd
        self.dem=abs(Dem)//gcd

    def __add__(self, other):
        return Fraction(self.sign*self.num*other.dem + other.sign*other.
↪num*self.dem, self.dem*other.dem)

    def __sub__(self, other):
        return Fraction(self.sign*self.num*other.dem - other.sign*other.
↪num*self.dem, self.dem*other.dem)

    def __mul__(self, other):
        return Fraction(self.sign*other.sign*self.num*other.num, self.dem*other.
↪dem)

    def __truediv__(self, other):
        return self * Fraction(other.sign*other.dem, other.num)

    def __gt__(self,other):
        if self.sign*self.num*other.dem > other.sign*other.num*self.dem:
            return True
        return False

    def __ge__(self,other):
        if self.sign*self.num*other.dem >= other.sign*other.num*self.dem:
            return True
        return False

    def __eq__(self, other):
        if self.sign*self.num*other.dem == other.sign*other.num*self.dem:
            return True
        return False

    def getNum(self):
        return self.num
    def getDem(self):
        return self.dem

    def __str__(self):
        return str(self.sign*self.num)+" // "+ str(self.dem)
```

3 Test działania

```
[3]: txt="""a=Fraction(1,2)
b=Fraction(1,3)

a+b
a-b
a*b
a/b

a>b
a<=b

a.getNum()
a.getDem

a=Fraction(-1,1)
b=Fraction(1,-1)
a==b"""
```

```
[4]: for line in txt.splitlines():
         if line:
             print(">>> "+line)
             exec("d="+line)
             print("  ",d)
             print()
         else:
             print()
```

```
>>> a=Fraction(1,2)
   1 // 2

>>> b=Fraction(1,3)
   1 // 3


>>> a+b
   5 // 6
```

```
>>> a-b
   1 // 6

>>> a*b
   1 // 6

>>> a/b
   3 // 2


>>> a>b
   True

>>> a<=b
   False


>>> a.getNum()
   1

>>> a.getDem
   <bound method Fraction.getDem of <__main__.Fraction object at
0x0000026FA7E4FD60>>


>>> a=Fraction(-1,1)
   -1 // 1

>>> b=Fraction(1,-1)
   -1 // 1

>>> a==b
   True
```

[5]:
```
c=Fraction(1/2,1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-3d25b2744a7b> in <module>
----> 1 c=Fraction(1/2,1)

<ipython-input-2-3c47de94392c> in __init__(self, Num, Dem)
      2     def __init__(self, Num, Dem):
      3         if type(Num)!=int or type(Dem)!= int:
----> 4             raise TypeError("Muszą być liczby całkowite")
      5
      6         if Dem==0:

TypeError: Muszą być liczby całkowite
```

4 Nadprogramowe

Dodatkowa klasa zawiera:

- Możliwość tworzenia ułamków z dowolnych liczb (np 0.5, math.pi)
- Dodawanie, mnożenie, odejmowanie, dzielenie klasy do innych typów liczbowych
- Potęgowanie
- Wyświetlanie w wersji mieszanej (zwykłej, jeśli część całkowita to 0) lub dziesiętnej
- Porównywanie z innymi typami danych
- Możliwość używania +-Frac() (k=-f)
- Działające funkcjie int, float, abs
- Działają z funkcjami matematycznymi (sin, cos, exp, ect.)

… i wiele więcej!

[6]:

```python
class Frac:

    mixed=False # Normalny czy mieszany
    precision=0 # Do którego miejsca po przecinku cyfry mają znaczenie. 0-maksynalne
    decimal=False # Czy wyświetlać w postaci ułamka dziesiętnego (ważniejsze niż mixed)

    def __init__(self, Num, Dem=1):

        if type(Num) not in (float,int, Frac) or type(Dem) not in (float,int,Frac):
            raise TypeError("Musisz podać liczbę")

        if Dem==0:
            raise ValueError("Mianownik musi być różny od zera")

        self.sign = 1 if Num*Dem>0 else -1 if Num*Dem<0 else 0


        if Frac.precision==0:
            num_temp=Num.as_integer_ratio()
            dem_temp=Dem.as_integer_ratio()
            num=abs(num_temp[0]*dem_temp[1])
            dem=abs(num_temp[1]*dem_temp[0])
        else:
            num=int(abs(Num*10**Frac.precision))
            dem=int(abs(Dem*10**Frac.precision))

        if num!=0:
            gcd=Greatest_common_divisor(num,dem)
        else:
            gcd=1
            dem=1
        self.num=num//gcd
        self.dem=dem//gcd

    def __add__(self, other):
        if type(other)!=Frac:
            other=Frac(other)
        return Frac(self.sign*self.num*other.dem + other.sign*other.num*self.dem, self.dem*other.dem)

    def __radd__(self, other):
        return self + other

    def __sub__(self, other):
        #return Frac(self.sign*self.num*other.dem - other.sign*other.num*self.dem, self.dem*other.dem)
        return self+(-1)*other

    def __rsub__(self, other):
        return -1*self+other

    def __pow__(self, power):
        if type(power)==Frac:
            power=power.num/power.dem
        return Frac((self.sign*self.num)**power, self.dem**power) if power > 0 else Frac((self.sign*self.dem)**(-power), self.num**(-power))

    def __rmul__(self, other):
        return self*other

    def __mul__(self, other):
        if type(other)!=Frac:
            other=Frac(other)
        return Frac(self.sign*other.sign*self.num*other.num, self.dem*other.dem)

    def __truediv__(self, other):
        return self * other**(-1)
    def __gt__(self,other):
        if type(other)!=Frac:
            other=Frac(other)
        if self.sign*self.num*other.dem > other.sign*other.num*self.dem:
            return True
        return False

    def __ge__(self,other):
        if type(other)!=Frac:
            other=Frac(other)
        if self.sign*self.num*other.dem >= other.sign*other.num*self.dem:
            return True
        return False

    def __eq__(self, other):
        if type(other)!=Frac:
            other=Frac(other)
        if self.sign*self.num*other.dem == other.sign*other.num*self.dem:
            return True
        return False

    def __lt__(self,other):
        if type(other)!=Frac:
            other=Frac(other)
        if self.sign*self.num*other.dem < other.sign*other.num*self.dem:
            return True
        return False

    def __le__(self,other):
        if type(other)!=Frac:
            other=Frac(other)
        if self.sign*self.num*other.dem <= other.sign*other.num*self.dem:
            return True
        return False

    def getNum(self):
        return self.num

    def getDem(self):
        return self.dem

    def __str__(self):
        if Frac.decimal:
            return str(self.sign*self.num/self.dem)
        elif not self.mixed or self.num//self.dem == 0:
            return str(self.sign*self.num)+" // "+ str(self.dem) if self.dem != 1 else str(self.sign*self.num)
        else:
            a = self.num//self.dem
            b = self.num - a*self.dem
            return str(self.sign*a) + " i " + str(b) + " // " + str(self.dem) if b !=0 else str(self.sign*a)

    def __repr__(self):
        return self.__str__()

    def __abs__(self):
        return Frac(self*self.sign)

    def as_integer_ratio(self):
        return (self.sign*self.num, self.dem)

    def __neg__(self):
        return Frac((-1)*self)

    def __pos__(self):
        return Frac(self)

    def __float__(self):
        return self.sign*self.num/self.dem

    def __int__(self):
        return self.sign*(self.num//self.dem)
```

5 Test

[7]:

```
import math
a=Frac(1,2)
b=Frac(1,3)
```

[8]:

```
print(a)
b
```

```
1 // 2
```

[8]:

```
1 // 3
```

[9]:

```
import math
txt="""
a=Frac(1,2)
b=Frac(1,3)
c=Frac(-1/2)
f=Frac(math.pi)
```

```
a+b
a-b
a*b
a+c
a-c
a*5
Frac.mixed=True
a*5
f
Frac.decimal=True
f
h=a**b
h
h=h**3
h
h-a
a-1/2
Frac(1/4)**a
Frac.decimal=False
Frac.mixed=False
a=Frac(1,1)
a
a/2
a>1/2
a/2>1/2
a/2>=1/2
1/2>=a/2
a/2>=Frac(1/3)
1==a
(a/2).as_integer_ratio()

Frac.decimal=True
k=-f
k
k+f

Frac.decimal=False


float(f)
int(f)
f
math.sin(f)
math.sin(math.pi)-math.sin(f)
math.exp(Frac(1,2))-math.e**(1/2)"""
```

[10]:

```
for line in txt.splitlines():
    if line:
        print(">>> "+line)
        exec("d="+line)
        print("  ",d)
        print()
    else:
        print()
```

```
>>> a=Frac(1,2)
   1 // 2

>>> b=Frac(1,3)
   1 // 3

>>> c=Frac(-1/2)
   -1 // 2

>>> f=Frac(math.pi)
   884279719003555 // 281474976710656


>>> a+b
   5 // 6

>>> a-b
   1 // 6

>>> a*b
   1 // 6

>>> a+c
   0

>>> a-c
   1

>>> a*5
   5 // 2

>>> Frac.mixed=True
   True

>>> a*5
   2 i 1 // 2
```

```
>>> f
   3 i 39854788871587 // 281474976710656

>>> Frac.decimal=True
   True

>>> f
   3.141592653589793

>>> h=a**b
   0.7937005259840997

>>> h
   0.7937005259840997

>>> h=h**3
   0.49999999999999994

>>> h
   0.49999999999999994

>>> h-a
   -3.0834472233596806e-17

>>> a-1/2
   0.0

>>> Frac(1/4)**a
   0.5

>>> Frac.decimal=False
   False

>>> Frac.mixed=False
   False

>>> a=Frac(1,1)
   1

>>> a
   1

>>> a/2
   1 // 2

>>> a>1/2
   True
```

```
>>> a/2>1/2
   False

>>> a/2>=1/2
   True

>>> 1/2>=a/2
   True

>>> a/2>=Frac(1/3)
   True

>>> 1==a
   True

>>> (a/2).as_integer_ratio()
   (1, 2)


>>> Frac.decimal=True
   True

>>> k=-f
   -3.141592653589793

>>> k
   -3.141592653589793

>>> k+f
   0.0


>>> Frac.decimal=False
   False



>>> float(f)
   3.141592653589793

>>> int(f)
   3

>>> f
   884279719003555 // 281474976710656

>>> math.sin(f)
   1.2246467991473532e-16
```

```
>>> math.sin(math.pi)-math.sin(f)
   0.0

>>> math.exp(Frac(1,2))-math.e**(1/2)
   0.0
```

[11]:

```
f=Frac(1,0)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-bc4f77eb7303> in <module>
----> 1 f=Frac(1,0)

<ipython-input-6-97308a871bc7> in __init__(self, Num, Dem)
     11 
     12         if Dem==0:
---> 13             raise ValueError("Mianownik musi być różny od zera")
     14 
     15         self.sign = 1 if Num*Dem>0 else -1 if Num*Dem<0 else 0

ValueError: Mianownik musi być różny od zera
```