Algorytmy i stuktury danych - Lista 1

Alicja Myśliwiec - gr. wtorek 7:30

In [166…

```python
from math import gcd
```

Dodatkowa funckja 'verify_one_type' ma na celu jedynie sprawdzenie, czy wartość na której chcemy operować jest obiektem klasy Fraction. Została wydzielona, żeby nie powtarzać ciągle kilku linijek kodu :)

In [189…

```python
def verify_one_type(value):
    """
    Function checks if the entered value type is compatible to Fraction class
    """

    if not isinstance(value, Fraction):
        raise TypeError("Invalid type! You can only operate with fractions")

    pass
```

In [190…

```python
class Fraction:
    """
    Class to make fractions
    """

    constant = 2

    @staticmethod
    def mixed(element):
        """
        It's a static method that allows us to set a fraction appearance function
        :param element: if 'True' we get mixed fraction, 'False' - simple
        """

        if element == "True":
            Fraction.constant = 3

        elif element == "False":
            Fraction.constant = 2

        else:
            raise TypeError("The argument must be 'True' or 'False'")

    def __init__(self, numerator, denominator):

        if denominator == 0:
            raise ZeroDivisionError("Cannot create a fraction with 0 as a denominator")

        if type(numerator) is not int or type(denominator) is not int:
            raise TypeError("Numerator and denominator must be integers")

        number = gcd(numerator, denominator)
        self.num = numerator // number
        self.den = denominator // number

        if self.num < 0 and self.den < 0 or self.den < 0:
            self.num *= -1
            self.den *= -1

        self.approx = self.num / self.den

    def __add__(self, other):
        """
        :return: result, fraction class object
        """

        verify_one_type(other)

        if self.den == other.den:

            new_den = self.den
            new_num = self.num + other.num

        else:

            new_den = self.den * other.den
            new_num = self.num * other.den + other.num * self.den

        return Fraction(new_num, new_den)

    def __sub__(self, other):
        """
        :return: result, fraction class object
        """

        verify_one_type(other)

        if self.den == other.den:

            new_den = self.den
            new_num = self.num - other.num

        else:

            new_den = self.den * other.den
            new_num = self.num * other.den - other.num * self.den

        return Fraction(new_num, new_den)

    def __mul__(self, other):
        """
        :return: result, fraction class object
        """

        verify_one_type(other)

        new_num = self.num * other.num
        new_den = self.den * other.den

        return Fraction(new_num, new_den)

    def __truediv__(self, other):
        """
        :return: result, fraction class object
        """

        verify_one_type(other)

        if other.num == 0:
            raise ZeroDivisionError("Cannot be divided by zero :c")

        new_num = int(self.num * other.den)
        new_den = int(self.den * other.num)

        return Fraction(new_num, new_den)

    def __neg__(self):

        new_num = self.num * -1

        return Fraction(new_num, self.den)

    def __lt__(self, other):

        verify_one_type(other)

        if self.approx < other.approx:
            return True
        else:
            return False

    def __gt__(self, other):

        verify_one_type(other)

        if self.approx > other.approx:
            return True
        else:
            return False

    def __le__(self, other):

        verify_one_type(other)

        if self.approx <= other.approx:
            return True
        else:
            return False

    def __ge__(self, other):

        verify_one_type(other)

        if self.approx >= other.approx:
            return True
        else:
            return False

    def __eq__(self, other):

        verify_one_type(other)

        if self.approx == other.approx:
            return True
        else:
            return False

    def __ne__(self, other):

        verify_one_type(other)

        if self.approx != other.approx:
            return True
        else:
            return False

    def __str__(self):
        """
        Function represents the appearance of the fraction
        :return: depending on how 'mixed' method is set, either simple or mixed fraction is returned
        """

        if abs(self.num) == abs(self.den) or self.den == 1:
            return "{}".format(int(self.num))

        if Fraction.constant == 3 and abs(self.num) > self.den:
            big_number = int(abs(self.num) // self.den)
            new_num = int(abs(self.num) % self.den)

            if self.num < 0:
                big_number *= -1

            return "{}({}/{})".format(big_number, new_num, self.den)

        else:

            return "{}/{}".format(int(self.num), int(self.den))

    def get_num(self):
        """
        :return: Numerator
        """

        return self.num

    def get_den(self):
        """
        :return: Denominator
        """

        return self.den
```

Ułamek domyślnie pokazuje się jako 'niewłaściwy' (Fraction.mixed("False"))

In [191…

```python
f1 = Fraction(1,4)
f2 = Fraction(1,2)
f3 = f1 + f2
print(f3)
```

```
3/4
```

In [192…

```python
f4 = Fraction(7, -5)
f5 = Fraction(9, 2)
f6 = Fraction(3, 7)
```

In [193…

```python
print(f4, f4.get_num(), f4.get_den())
```

```
-7/5 -7 5
```

In [194…

```python
f4 > f5
```

Out[194…

```
False
```

In [195…

```python
f5 == f6
```

Out[195…

```
False
```

In [196…

```python
f6 >= f4
```

Out[196…

```
True
```

Funckją mixed można zmienić sposób wyświetlania ułamków jako 'mieszane'

In [197…

```python
Fraction.mixed("True")
```

In [198…

```python
print(f4)
```

```
-1(2/5)
```

Przeróżne sytuacje warte pokazania:

In [199…

```python
Fraction(1, 0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-199-8d47e7ec56c0> in <module>
----> 1 Fraction(1, 0)

<ipython-input-190-3dbb8dbbbf8c> in __init__(self, numerator, denominator)
     25 
     26         if denominator == 0:
---> 27             raise ZeroDivisionError("Cannot create a fraction with 0 as a denominator")
     28 
     29         if type(numerator) is not int or type(denominator) is not int:

ZeroDivisionError: Cannot create a fraction with 0 as a denominator
```

In [200…

```python
Fraction(1.5, 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-200-006e5ad5f25d> in <module>
----> 1 Fraction(1.5, 3)

<ipython-input-190-3dbb8dbbbf8c> in __init__(self, numerator, denominator)
     28 
     29         if type(numerator) is not int or type(denominator) is not int:
---> 30             raise TypeError("Numerator and denominator must be integers")
     31 
     32         number = gcd(numerator, denominator)

TypeError: Numerator and denominator must be integers
```

In [201…

```python
print(Fraction(0, 3))
```

```
0
```

In [202…

```python
Fraction(1, 2) / Fraction(0, 3)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-202-59826b24243b> in <module>
----> 1 Fraction(1, 2) / Fraction(0, 3)

<ipython-input-190-3dbb8dbbbf8c> in __truediv__(self, other)
     98 
     99         if other.num == 0:
--> 100             raise ZeroDivisionError("Cannot be divided by zero :c")
    101 
    102         new_num = int(self.num * other.den)

ZeroDivisionError: Cannot be divided by zero :c
```

In [203…

```python
Fraction(1, 2) + 'a'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-203-0ceca7ade453> in <module>
----> 1 Fraction(1, 2) + 'a'

<ipython-input-190-3dbb8dbbbf8c> in __add__(self, other)
     45         """
     46 
---> 47         verify_one_type(other)
     48 
     49         if self.den == other.den:

<ipython-input-189-10e5aaf79d92> in verify_one_type(value)
      5 
      6     if not isinstance(value, Fraction):
----> 7         raise TypeError("Invalid type! You can only operate with fractions")
      8 
      9     pass

TypeError: Invalid type! You can only operate with fractions
```

Efekt skracania ułamków i przejrzystej prezentacji minusów:

In [204…

```python
print(Fraction(6, 2))
```

```
3
```

In [205…

```python
print(Fraction(-4, -1))
```

```
4
```

In [206…

```python
print(Fraction(3, -7))
```

```
-3/7
```

Podstawowe działania na ułamkach:

In [207…

```python
f7 = f4*f6-f5
print(f7)
```

```
-5(1/10)
```

In [208…

```python
Fraction.mixed("False")
```

In [209…

```python
print(-f5+f4)
```

```
-59/10
```

In [210…

```python
f8 = f6 / f1
print(f8)
```

```
12/7
```

LINK GITHUB

https://github.com/AlutkaMalutka/Programowanie_python/tree/main/semestr_3/Lista_1-3s-