МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №4.2 по дисциплине основы программной инженерии

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2022 г.

Выполнение:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError("Illegal value of the denominator")
        self.__numerator = a
        self.__denominator = b
        self.__reduce()

    # Сокращение дроби.
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            if a == 0:
                return b
            elif b == 0:
                return a
            elif a >= b:
                return gcd(a % b, b)
            else:
                return gcd(a, b % a)

        sign = 1
        if (self.__numerator > 0 > self.__denominator) or \
                (self.__numerator < 0 < self.__denominator):
            sign = -1

        a, b = abs(self.__numerator), abs(self.__denominator)
        c = gcd(a, b)
        self.__numerator = sign * (a // c)
        self.__denominator = b // c

    # Клонировать дробь.
    def __clone(self):
        return Rational(self.__numerator, self.__denominator)

    @property
    def numerator(self):
        return self.__numerator

    @numerator.setter
    def numerator(self, value):
        self.__numerator = int(value)
        self.__reduce()

    @property
    def denominator(self):
        return self.__denominator

    @denominator.setter
    def denominator(self, value):
        value = int(value)
        if value == 0:
            raise ValueError("Illegal value of the denominator")
        self.__denominator = value
        self.__reduce()
```

```
    # Привести дробь к строке.
    def __str__(self):
        return f"{self.__numerator} / {self.__denominator}"

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

    # Привести дробь к вещественному значению.
    def __float__(self):
        return self.__numerator / self.__denominator

    # Привести дробь к логическому значению.
    def __bool__(self):
        return self.__numerator != 0

    # Сложение обыкновенных дробей.
    def __iadd__(self, rhs):  # +=
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator + \
                self.denominator * rhs.numerator
            b = self.denominator * rhs.denominator
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __add__(self, rhs):  # +
        return self.__clone().__iadd__(rhs)

    # Вычитание обыкновенных дробей.
    def __isub__(self, rhs):  # -=
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator - \
                self.denominator * rhs.numerator
            b = self.denominator * rhs.denominator
            self.__numerator, self.__denominator = a, b

            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __sub__(self, rhs):  # -
        return self.__clone().__isub__(rhs)

    # Умножение обыкновенных дробей.
    def __imul__(self, rhs):  # *=
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.numerator
            b = self.denominator * rhs.denominator
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __mul__(self, rhs):  # *
        return self.__clone().__imul__(rhs)

    # Деление обыкновенных дробей.
    def __itruediv__(self, rhs):  # /=
        if isinstance(rhs, Rational):
```

```
            a = self.numerator * rhs.denominator
            b = self.denominator * rhs.numerator
            if b == 0:
                raise ValueError("Illegal value of the denominator")
            self.__numerator, self.__denominator = a, b
            self.__reduce()
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __truediv__(self, rhs):  # /
        return self.__clone().__itruediv__(rhs)

    # Отношение обыкновенных дробей.
    def __eq__(self, rhs):  # ==
        if isinstance(rhs, Rational):
            return (self.numerator == rhs.numerator) and \
                   (self.denominator == rhs.denominator)
        else:
            return False

    def __ne__(self, rhs):  # !=
        if isinstance(rhs, Rational):
            return not self.__eq__(rhs)
        else:
            return False

    def __gt__(self, rhs):  # >
        if isinstance(rhs, Rational):
            return self.__float__() > rhs.__float__()
        else:
            return False

    def __lt__(self, rhs):  # <
        if isinstance(rhs, Rational):
            return self.__float__() < rhs.__float__()
        else:
            return False

    def __ge__(self, rhs):  # >=
        if isinstance(rhs, Rational):
            return not self.__lt__(rhs)
        else:
            return False

    def __le__(self, rhs):  # <=
        if isinstance(rhs, Rational):
            return not self.__gt__(rhs)
        else:
            return False


if __name__ == '__main__':
    r1 = Rational(3, 4)
    print(f"r1 = {r1}")
    r2 = Rational(5, 6)
    print(f"r2 = {r2}")
    print(f"r1 + r2 = {r1 + r2}")
    print(f"r1 - r2 = {r1 - r2}")
    print(f"r1 * r2 = {r1 * r2}")
    print(f"r1 / r2 = {r1 / r2}")
    print(f"r1 == r2: {r1 == r2}")
    print(f"r1 != r2: {r1 != r2}")
    print(f"r1 > r2: {r1 > r2}")
```

```
    print(f"r1 < r2: {r1 < r2}")
    print(f"r1 >= r2: {r1 >= r2}")
    print(f"r1 <= r2: {r1 <= r2}")
```

Пример 1

```
r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r1 == r2: False
r1 != r2: True
r1 > r2: False
r1 < r2: True
r1 >= r2: False
r1 <= r2: True

Process finished with exit code 0
```

Результат работы примера 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


class Money:

    def   init  (
            self,
            penny_1=0,
            penny_5=0,
            penny_10=0,
            penny_50=0,
            ruble_1=0,
            ruble_2=0,
            ruble_5=0,
            ruble_10=0,
            ruble_50=0,
            ruble_100=0,
            ruble_200=0,
            ruble_500=0,
            ruble_1000=0,
            ruble_2000=0,
            ruble_5000=0
    ):
        self.__penny_1 = int(penny_1)
        self.__penny_5 = int(penny_5)
        self.__penny_10 = int(penny_10)
        self.__penny_50 = int(penny_50)

        self.__ruble_1 = int(ruble_1)
        self.__ruble_2 = int(ruble_2)
        self.__ruble_5 = int(ruble_5)
        self.__ruble_10 = int(ruble_10)
```

```
        self.__ruble_50 = int(ruble_50)
        self.__ruble_100 = int(ruble_100)
        self.__ruble_200 = int(ruble_200)
        self.__ruble_500 = int(ruble_500)
        self.__ruble_1000 = int(ruble_1000)
        self.__ruble_2000 = int(ruble_2000)
        self.__ruble_5000 = int(ruble_5000)

        self.__set_summa()

    def __del__(self):
        print(f"{self.summa} удалено")

    def __str__(self):
        return f"{self.summa}"



    @property
    def penny_1(self):
        return self.__penny_1

    @property
    def penny_5(self):
        return self.__penny_5

    @property
    def penny_10(self):
        return self.__penny_10

    @property
    def penny_50(self):
        return self.__penny_50

    @property
    def ruble_1(self):
        return self.__ruble_1

    @property
    def ruble_2(self):
        return self.__ruble_2

    @property
    def ruble_5(self):
        return self.__ruble_5

    @property
    def ruble_10(self):
        return self.__ruble_10

    @property
    def ruble_50(self):
        return self.__ruble_50

    @property
    def ruble 100(self):
        return self.__ruble_100

    @property
    def ruble_200(self):
        return self.__ruble_200

    @property
    def ruble_500(self):
```

```
        return self.__ruble_500

    @property
    def ruble_1000(self):
        return self.__ruble_1000

    @property
    def ruble_2000(self):
        return self.__ruble_2000

    @property
    def ruble_5000(self):
        return self.__ruble_5000

    @property
    def summa(self):
        return self.__summa

    def __set_summa(self):
        summa = \
            self.penny_1 * 0.01 + \
            self.penny_5 * 0.05 + \
            self.penny_10 * 0.1 + \
            self.penny_50 * 0.5 + \
            self.ruble_1 * 1 + \
            self.ruble_2 * 2 + \
            self.ruble_5 * 5 + \
            self.ruble_10 * 10 + \
            self.ruble_50 * 50 + \
            self.ruble_100 * 100 + \
            self.ruble_200 * 200 + \
            self.ruble_500 * 500 + \
            self.ruble_1000 * 1000 + \
            self.ruble_2000 * 2000 + \
            self.ruble_5000 * 5000
        self.__summa = float(summa)

    def change_amount(
            self,
            penny_1=-1,
            penny_5=-1,
            penny_10=-1,
            penny_50=-1,
            ruble_1=-1,
            ruble_2=-1,
            ruble_5=-1,
            ruble_10=-1,
            ruble_50=-1,
            ruble_100=-1,
            ruble_200=-1,
            ruble_500=-1,
            ruble_1000=-1,
            ruble_2000=-1,
            ruble_5000=-1
    ):
        if penny_1 > 0:
            self.__penny_1 = penny_1
        if penny_5 > 0:
            self.__penny_5 = penny_5
        if penny_10 > 0:
            self.__penny_10 = penny_10
        if penny_50 > 0:
            self.__penny_50 = penny_50
```

```
        if ruble_1 > 0:
            self.__ruble_1 = ruble_1
        if ruble_2 > 0:
            self.__ruble_2 = ruble_2
        if ruble_5 > 0:
            self.__ruble_5 = ruble_5
        if ruble_10 > 0:
            self.__ruble_10 = ruble_10
        if ruble_50 > 0:
            self.__ruble_50 = ruble_50
        if ruble_100 > 0:
            self.__ruble_100 = ruble_100
        if ruble_200 > 0:
            self.__ruble_200 = ruble_200
        if ruble_500 > 0:
            self.__ruble_500 = ruble_500
        if ruble_1000 > 0:
            self.__ruble_1000 = ruble_1000
        if ruble_2000 > 0:
            self.__ruble_2000 = ruble_2000
        if ruble_5000 > 0:
            self.__ruble_5000 = ruble_5000
        self.__set_summa()

    def count_summa(self, second):
        if isinstance(second, Money):
            return self.summa + second.summa
        return None

    def __add__(self, other):
        if isinstance(other, Money):
            return self.summa + other.summa
        return None

    def count_difference(self, second):
        if isinstance(second, Money):
            return self.summa - second.summa
        return None

    def __sub__(self, other):
        if isinstance(other, Money):
            return self.summa - other.summa
        return None

    def count_division(self, second):
        if isinstance(second, Money):
            if second.summa != 0:
                return self.summa / second.summa
        return None

    def float_division(self, number):
        if isinstance(number, float):
            return self.summa / number
        return None

    def   truediv  (self, other):
        if isinstance(other, Money):
            if other.summa != 0:
                return self.summa / other.summa
        if isinstance(other, float):
            if other != 0:
                return self.summa / other
        return None
```

```
    def float_multiplication(self, number):
        if isinstance(number, float):
            return self.summa * number
        return None

    def __mul__(self, other):
        if isinstance(other, Money):
            return self.summa * other.summa
        if isinstance(other, float):
            return self.summa * other
        return None

    def compare(self, second):
        if isinstance(second, Money):
            return self.summa > second.summa
        return None

    def __lt__(self, other):
        if isinstance(other, Money):
            if self.summa < other.summa:
                return True
            else:
                return False
        return None

    def __le__(self, other):
        if isinstance(other, Money):
            if self.summa <= other.summa:
                return True
            else:
                return False
        return None

    def __eq__(self, other):
        if isinstance(other, Money):
            if self.summa == other.summa:
                return True
            else:
                return False
        return None

    def __ne__(self, other):
        if isinstance(other, Money):
            if self.summa != other.summa:
                return True
            else:
                return False
        return None

    def __gt__(self, other):
        if isinstance(other, Money):
            if self.summa > other.summa:
                return True
            else:
                return False
        return None

    def __ge__(self, other):
        if isinstance(other, Money):
            if self.summa >= other.summa:
                return True
            else:
                return False
        return None
```

```
    def display(self):
        print(f"Количество монет с достоинством 1 копейка: {self.penny_1}")
        print(f"Количество монет с достоинством 5 копеек: {self.penny_5}")
        print(f"Количество монет с достоинством 10 копеек: {self.penny_10}")
        print(f"Количество монет с достоинством 50 копеек: {self.penny_50}")

        print(f"Количество монет с достоинством 1 рубль: {self.ruble_1}")
        print(f"Количество монет с достоинством 2 рубля: {self.ruble_2}")
        print(f"Количество монет с достоинством 5 рублей: {self.ruble_5}")
        print(f"Количество монет с достоинством 10 рублей: {self.ruble_10}")
        print(f"Количество купюр с достоинством 50 рублей: {self.ruble_50}")
        print(f"Количество купюр с достоинством 100 рублей:
{self.ruble_100}")
        print(f"Количество купюр с достоинством 200 рублей:
{self.ruble_200}")
        print(f"Количество купюр с достоинством 500 рублей:
{self.ruble_500}")
        print(f"Количество купюр с достоинством 1000 рублей:
{self.ruble_1000}")
        print(f"Количество купюр с достоинством 2000 рублей:
{self.ruble_2000}")
        print(f"Количество купюр с достоинством 5000 рублей:
{self.ruble_5000}")
        print(f"Общая сумма: {self.summa}")

    def read(self):
        penny_1 = int(input("Введите количество монет с достоинством 1
копейка: "))
        penny_5 = int(input("Введите количество монет с достоинством 5
копеек: "))
        penny_10 = int(input("Введите количество монет с достоинством 10
копеек: "))
        penny_50 = int(input("Введите количество монет с достоинством 50
копеек: "))

        ruble_1 = int(input("Введите количество монет с достоинством 1 рубль:
"))
        ruble_2 = int(input("Введите количество монет с достоинством 2 рубля:
"))
        ruble_5 = int(input("Введите количество монет с достоинством 5
рублей: "))
        ruble_10 = int(input("Введите количество монет с достоинством 10
рублей: "))
        ruble_50 = int(input("Введите количество купюр с достоинством 50
рублей: "))
        ruble_100 = int(input("Введите количество купюр с достоинством 100
рублей: "))
        ruble_200 = int(input("Введите количество купюр с достоинством 200
рублей: "))
        ruble_500 = int(input("Введите количество купюр с достоинством 500
рублей: "))
        ruble_1000 = int(input("Введите количество купюр с достоинством 1000
рублей: "))
        ruble_2000 = int(input("Введите количество купюр с достоинством 2000
рублей: "))
        ruble_5000 = int(input("Введите количество купюр с достоинством 5000
рублей: "))

        self.__penny_1 = int(penny_1)
        self.__penny_5 = int(penny_5)
        self.__penny_10 = int(penny_10)
        self.__penny_50 = int(penny_50)
```

```
        self.__ruble_1 = int(ruble_1)
        self.__ruble_2 = int(ruble_2)
        self.__ruble_5 = int(ruble_5)
        self.__ruble_10 = int(ruble_10)
        self.__ruble_50 = int(ruble_50)
        self.__ruble_100 = int(ruble_100)
        self.__ruble_200 = int(ruble_200)
        self.__ruble_500 = int(ruble_500)
        self.__ruble_1000 = int(ruble_1000)
        self.__ruble_2000 = int(ruble_2000)
        self.__ruble_5000 = int(ruble_5000)

        self.__set_summa()


if __name__ == "__main__":
    m1 = Money(
        penny_1=10,
        penny_5=5,
        penny_10=15,
        penny_50=31,
        ruble_1=24,
        ruble_2=10,
        ruble_5=5,
        ruble_10=10,
        ruble_50=2,
        ruble_100=14,
        ruble_200=10,
        ruble_500=5,
        ruble_1000=20,
        ruble_2000=2,
        ruble_5000=1
    )
    m1.display()
    print(m1.float_division(2.5))
    print(m1)
    print(m1 / 2.5)
    print(m1.float_multiplication(0.5))
    print(m1 * 0.5)
    m2 = Money()
    m2.read()
    print(m1.compare(m2))
    print(m1 > m2)
    print(m1 >= m2)
    print(m1 < m2)
    print(m1 <= m2)
    print(m1 == m2)
    print(m1 != m2)
    print(m1.count_division(m2))
    print(m1 / m2)
    print(m2.count_difference(m1))
    print(m2 - m1)
    print(m2.count_summa(m1))
    print(m2 + m1)
    m2.display()
    m2.change amount(ruble 5=2)
    m2.display()
```

Решение первой индивидуальной задачи

Количество монет с достоинством 1 копейка: 10
Количество монет с достоинством 5 копеек: 5

Количество монет с достоинством 10 копеек: 15
Количество монет с достоинством 50 копеек: 31
Количество монет с достоинством 1 рубль: 24
Количество монет с достоинством 2 рубля: 10
Количество монет с достоинством 5 рублей: 5
Количество монет с достоинством 10 рублей: 10
Количество купюр с достоинством 50 рублей: 2
Количество купюр с достоинством 100 рублей: 14
Количество купюр с достоинством 200 рублей: 10
Количество купюр с достоинством 500 рублей: 5
Количество купюр с достоинством 1000 рублей: 20
Количество купюр с достоинством 2000 рублей: 2
Количество купюр с достоинством 5000 рублей: 1
Общая сумма: 35186.35
14074.539999999999
35186.35
14074.539999999999
17593.175
17593.175
Введите количество монет с достоинством 1 копейка: 21
Введите количество монет с достоинством 5 копеек: 45
Введите количество монет с достоинством 10 копеек: 12
Введите количество монет с достоинством 50 копеек: 56
Введите количество монет с достоинством 1 рубль: 12
Введите количество монет с достоинством 2 рубля: 56
Введите количество монет с достоинством 5 рублей: 23
Введите количество монет с достоинством 10 рублей: 65
Введите количество купюр с достоинством 50 рублей: 2
Введите количество купюр с достоинством 100 рублей: 6
Введите количество купюр с достоинством 200 рублей: 2
Введите количество купюр с достоинством 500 рублей: 56
Введите количество купюр с достоинством 1000 рублей: 2
Введите количество купюр с достоинством 2000 рублей: 7
Введите количество купюр с достоинством 5000 рублей: 3
False
False
False
True
True
False
True
0.5766301118342542
0.5766301118342542
25834.310000000005
25834.310000000005

96207.01000000001
96207.01000000001
Количество монет с достоинством 1 копейка: 21
Количество монет с достоинством 5 копеек: 45
Количество монет с достоинством 10 копеек: 12
Количество монет с достоинством 50 копеек: 56
Количество монет с достоинством 1 рубль: 12
Количество монет с достоинством 2 рубля: 56
Количество монет с достоинством 5 рублей: 23
Количество монет с достоинством 10 рублей: 65
Количество купюр с достоинством 50 рублей: 2
Количество купюр с достоинством 100 рублей: 6
Количество купюр с достоинством 200 рублей: 2
Количество купюр с достоинством 500 рублей: 56
Количество купюр с достоинством 1000 рублей: 2
Количество купюр с достоинством 2000 рублей: 7
Количество купюр с достоинством 5000 рублей: 3
Общая сумма: 61020.66
Количество монет с достоинством 1 копейка: 21
Количество монет с достоинством 5 копеек: 45
Количество монет с достоинством 10 копеек: 12
Количество монет с достоинством 50 копеек: 56
Количество монет с достоинством 1 рубль: 12
Количество монет с достоинством 2 рубля: 56
Количество монет с достоинством 5 рублей: 2
Количество монет с достоинством 10 рублей: 65
Количество купюр с достоинством 50 рублей: 2
Количество купюр с достоинством 100 рублей: 6
Количество купюр с достоинством 200 рублей: 2
Количество купюр с достоинством 500 рублей: 56
Количество купюр с достоинством 1000 рублей: 2
Количество купюр с достоинством 2000 рублей: 7
Количество купюр с достоинством 5000 рублей: 3
Общая сумма: 60915.66
35186.35 удалено
60915.66 удалено

Результата работы программы

```
class Money:

    def __init__(self, ruble, penny):
        self.__size = 100
        self.__value = [0 for _ in range(100)]
        self.__length = 0
        self.set_ruble(ruble)
        self.set_penny(penny)

    @property
```

```
    def size(self):
        return self.__size

    @property
    def value(self):
        return self.__value

    def set_ruble(self, value):
        if isinstance(value, int):
            temp = [0 for _ in range(98)]
            self.__value[2:100] = temp
            rubles = int(value)
            i = 2
            while rubles > 0:
                self.__value[i] = rubles % 10
                rubles = rubles // 10
                i += 1
            self.__length = i

    def set_penny(self, value):
        if isinstance(value, int):
            temp = [0 for _ in range(2)]
            self.__value[0:2] = temp
            penny = int(value)
            i = 0
            penny = penny % 100
            while penny > 0:
                self.__value[i] = penny % 10
                penny = penny // 10
                i += 1

    def __str__(self):
        temp = ""
        marker = False
        for i in reversed(self.__value):
            if 0 <= i <= 9:
                if 1 <= i <= 9:
                    marker = True
                if marker:
                    temp = temp + str(i)
        penny = temp[-2:]
        ruble = temp[0:len(temp) - 2]
        return f"{ruble}.{penny}"

    def __len__(self):
        return self.__length

    def __compare(self, second):
        if self.__length > len(second):
            return 1
        if self.__length < len(second):
            return 2
        if self.__length == len(second):
            for i in range(len(second) - 1, -1, -1):
                if self.value[i] > second.value[i]:
                    return 1
                if self.value[i] < second.value[i]:
                    return 2
        return 3

    def __lt__(self, other):
        if isinstance(other, Money):
            info = self.__compare(other)
            if info == 1:
```

```
                return True
            else:
                return False
        raise ValueError

    def __le__(self, other):
        if isinstance(other, Money):
            info = self.__compare(other)
            if info == 1 or info == 3:
                return True
            else:
                return False
        raise ValueError

    def __eq__(self, other):
        if isinstance(other, Money):
            info = self.__compare(other)
            if info == 3:
                return True
            else:
                return False
        raise ValueError

    def __ne__(self, other):
        if isinstance(other, Money):
            info = self.__compare(other)
            if info == 3:
                return False
            else:
                return True
        raise ValueError

    def __gt__(self, other):
        if isinstance(other, Money):
            info = self.__compare(other)
            if info == 2:
                return True
            else:
                return False
        raise ValueError

    def __ge__(self, other):
        if isinstance(other, Money):
            info = self.__compare(other)
            if info == 2 or info == 3:
                return True
            else:
                return False
        raise ValueError

    def __add__(self, other):
        if isinstance(other, Money):
            info = []
            ost = 0
            for i in range(100):
                ost = self.value[i] + other.value[i] + ost
                info.append(ost % 10)
                ost = ost // 10
            temp = ""
            marker = False
            for i in reversed(info):
                if 0 <= i <= 9:
                    if 1 <= i <= 9:
                        marker = True
```

```
                    if marker:
                        temp = temp + str(i)
            penny = temp[-2:]
            ruble = temp[0:len(temp) - 2]
            return f"{ruble}.{penny}"

    def __sub__(self, other):
        if isinstance(other, Money):
            info = []

            d1 = self.value.copy()
            d2 = other.value.copy()

            result = self.__compare(other)

            if result == 1:
                sign = 1
            if result == 2:
                sign = -1
            if result == 3:
                return 0

            if sign == 1:
                for i in range(100):
                    if d1[i] >= d2[i]:
                        info.append(d1[i] - d2[i])
                    else:
                        info.append(10 + d1[i] - d2[i])
                        if i != 99:
                            d1[i + 1] -= 1
            else:
                for i in range(100):
                    if d2[i] >= d1[i]:
                        info.append(d2[i] - d1[i])
                    else:
                        info.append(10 + d2[i] - d1[i])
                        if i != 99:
                            d2[i + 1] -= 1
            temp = ""
            marker = False
            for i in reversed(info):
                if 0 <= i <= 9:
                    if 1 <= i <= 9:
                        marker = True
                    if marker:
                        temp = temp + str(i)
            penny = temp[-2:]
            ruble = temp[0:len(temp) - 2]
            if sign == 1:
                return f"{ruble}.{penny}"
            if sign == -1:
                return f"-{ruble}.{penny}"


if __name__ == '__main__':
    m1 = Money(100, 20)
    m1.set_ruble(200)
    m1.set_penny(54)
    m2 = Money(400, 52)
    print(m1 > m2)
    print(m1 >= m2)
    print(m1 == m2)
    print(m1 != m2)
    print(m1 <= m2)
```

```
print(m1 < m2)
print(m1 - m2)
print(m2 + m1)
```

Решение второй индивидуальной задачи

```
True
True
False
True
False
False
-199.98
601.06

Process finished with exit code 0
```

Результат работы программы

Ответы на вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Перегрузка операторов — один из способов реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе.

__<название метода для переопределения>__

2. Какие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

__add__(self, other) - сложение. x + y вызывает x.__add__(y) .
__sub__(self, other) - вычитание (x - y).
__mul__(self, other) - умножение (x * y).
__truediv__(self, other) - деление (x / y).
__floordiv__(self, other) - целочисленное деление (x // y).
__mod__(self, other) - остаток от деления (x % y).
__divmod__(self, other) - частное и остаток (divmod(x, y)).
__pow__(self, other[, modulo]) - возведение в степень (x ** y , pow(x, y[, modulo])).
__lshift__(self, other) - битовый сдвиг влево (x << y).
__rshift__(self, other) - битовый сдвиг вправо (x >> y).
__and__(self, other) - битовое И (x & y).
__xor__(self, other) - битовое ИСКЛЮЧАЮЩЕЕ ИЛИ (x ^ y).
__or__(self, other) - битовое ИЛИ (x | y).
__lt__(self, other) - x < y вызывает x.__lt__(y) .
__le__(self, other) - x ≤ y вызывает x.__le__(y) .
__eq__(self, other) - x == y вызывает x.__eq__(y) .
__ne__(self, other) - x != y вызывает x.__ne__(y) .
__gt__(self, other) - x > y вызывает x.__gt__(y) .

__ge__(self, other) - x ≥ y вызывает x.__ge__(y) .

3. В каких случаях будут вызваны следующие методы: __add__ , __iadd__ и __radd__ ?

Приведите примеры.

__add__(self, other) - сложение. x + y вызывает x.__add__(y) .

__iadd__(self, other) - += .

x.__add__(y) , и только в том случае, если это не получилось, будет пытаться вызвать y.__radd__(x) . Аналогично для остальных методов.

4. Для каких целей предназначен метод __new__ ? Чем он отличается от метода __init__ ?

__new__(cls[, ...]) — управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу __init__ .

__init__(self[, ...]) - конструктор.

5. Чем отличаются методы __str__ и __repr__ ?

__repr__(self) - вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.

__str__(self) - вызывается функциями str, print и format. Возвращает строковое представление объекта.