Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ

По лабораторной работе №2

Дисциплины «Объектно-ориентированное программирование»

Выполнил:

Пустяков Андрей Сергеевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента цифровых и робототехнических систем и электроники института перспективной инженерии

_____

(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Перегрузка операторов в языке Python.

Цель: приобрести навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git flow» (рис. 1).

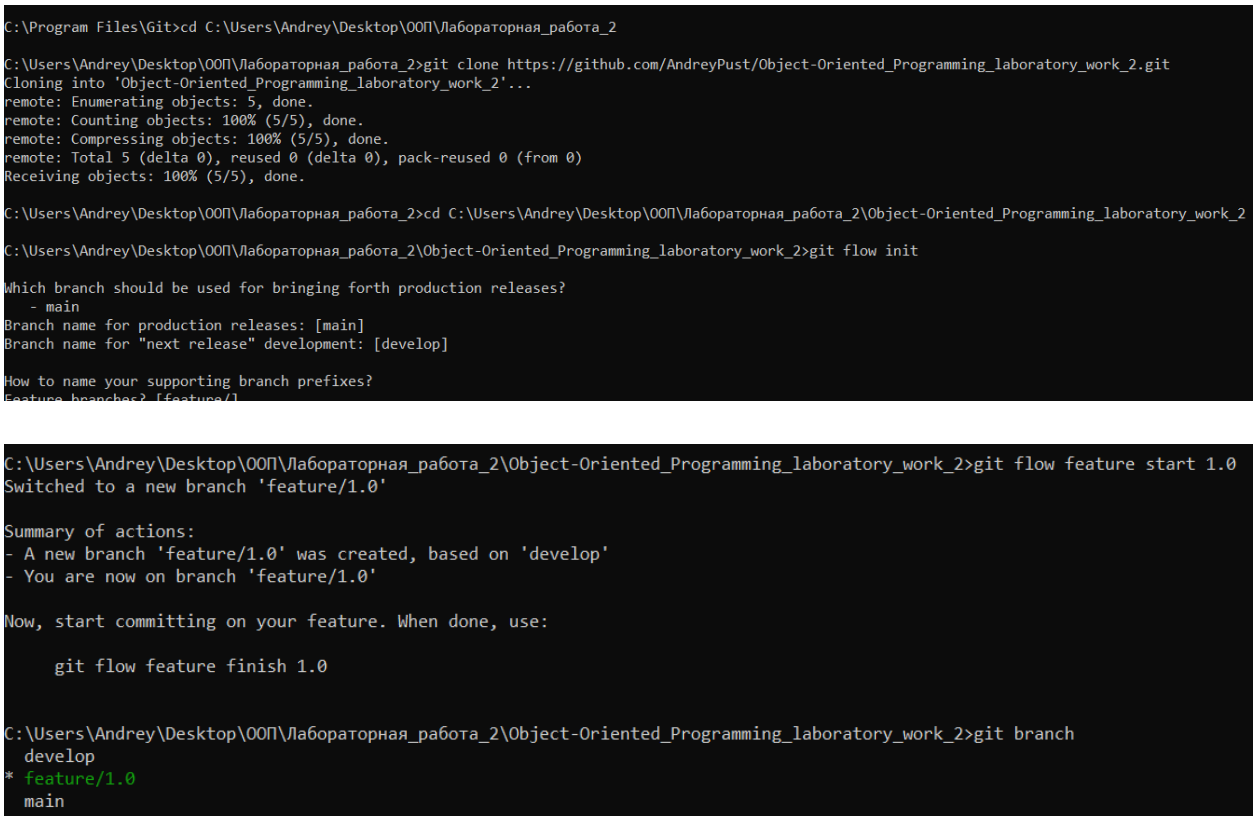

Рисунок 1 – Создание репозитория

Проработка примеров лабораторной работы:

Пример 1.

Необходимо изменить класс «Rational» из примера лабораторной работы 4.1, используя перегрузку операторов.

Код программы примера 1 с использованием перегрузки операторов:

```
#!/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 and self.__denominator < 0) or \
                (self.__numerator < 0 and self.__denominator > 0):
            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}")
```

Результаты работы данного кода, реализация арифметических операций и операций сравнения (рис. 2).

```
C:\Users\Andrey\anaconda3\envs\oop_2\python.exe C:\Users\Andrey\Desktop\
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
```

Рисунок 2 – Результаты работы кода примера 1

Выполнение индивидуальных заданий:

Вариант 25

Задание 1.

Необходимо выполнить индивидуальное задание 1 лабораторной работы 4.1, но максимально задействовав имеющиеся в Python средства перегрузки операторов.

Парой называется класс с двумя полями, которые обычно имеют имена «first» и «second». Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

– метод инициализации «__init__()» (метод должен контролировать значения аргументов на корректность);

– ввод с клавиатуры «read»;

– вывод на экран «display».

Реализовать внешнюю функцию с именем «make_тип()» , где тип – тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

Поле «first» – дробное положительное число, цена товара; поле «second» – целое положительное число, количество единиц товара. Реализовать метод «cost()» — вычисление стоимости товара (Вариант 25(5)).

Код программы решения индивидуального задания 1:

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

# Задание 1.
# Поле «first» - дробное положительное число, цена товара;
# Поле «second» - целое положительное число, количество единиц товара.
# Реализовать метод «cost()» — вычисление стоимости товара (Вариант 25(5)).
# Также необходимо максимально задействовать средства перегрузки операторов.

class Pair:
    def __init__(self, first, second):
        """
        Метод инициализации, проверяет аргументы на корректность.
        """
        if not isinstance(first, (int, float)) or first <= 0:
            raise ValueError("Цена товара должна быть положительным дробным
числом!")
        if not isinstance(second, int) or second <= 0:
            raise ValueError("Количество товара должно быть положительным
целым числом!")

        self.first = float(first)  # Цена товара
        self.second = int(second)  # Количество товара

    def read(self):
        """
        Ввод значений с клавиатуры.
        """
        try:
            self.first = float(input("Введите цену товара (положительное
дробное число): "))
            if self.first <= 0:
                raise ValueError("Цена должна быть положительным числом!")

            self.second = int(input("Введите количество товара (положительное
целое число): "))
            if self.second <= 0:
                raise ValueError("Количество должно быть положительным целым
```

```
числом!")
        except ValueError as e:
            print(f"Ошибка ввода: {e}")
            return False
        return True

    def display(self):
        """
        Вывод информации о товаре.
        """
        print(f"Цена товара: {self.first}, Количество: {self.second}")

    def cost(self):
        """
        Метод для вычисления стоимости товара.
        """
        return self.first * self.second

    # Перегрузка оператора сложения
    def __add__(self, other):
        """
        Переопределим метод для получения товара со средними характеристиками
двух товаров.
        """
        if not isinstance(other, Pair):
            raise TypeError("Сложение возможно только между товарами!")

        # Общее количество товаров
        total_quantity = self.second + other.second

        # Общая стоимость всех товаров
        total_cost = (self.first * self.second) + (other.first *
other.second)

        # Среднее количество товара
        average_quantity = total_quantity / 2

        # Средняя стоимость единицы товара
        average_price = total_cost / total_quantity

        # Создание нового объекта со "средними" характеристиками
        return Pair(average_price, round(average_quantity))

    # Перегрузка оператора вычитания
    def __sub__(self, other):
        """
        Теперь логика вычитания такова, что вычисляется количество первого
товара со своей стоимостью такая,
        как если бы не существовало второго товара (перераспределение всех
денег на первый товар).
        self - это товар, на который уйдут все деньги второго товара other.
        """
        if not isinstance(other, Pair):
            raise ValueError("Вычитать можно только товары!")

        # Общая стоимость обоих товаров
        total_cost = (self.first * self.second) + (other.first *
other.second)

        # Новое количество первого товара
        new_quantity = int(total_cost / self.first)

        # Создание нового объекта с вычисленным количеством
        return Pair(self.first, new_quantity)
```

```
    # Перегрузка оператора умножения
    def __mul__(self, number):
        """
        При умножении товара на число умножается только его количество.
        """
        if not isinstance(number, (int, float)) or number <= 0:
            raise ValueError("Умножение товара возможно только на
положительное число!")
        return Pair(self.first, self.second * number)

    # Перегрузка оператора деления (деление на число)
    def __truediv__(self, number):
        """
        При делении товара на число, его количество уменьшается в некоторое
количество раз.
        """
        if not isinstance(number, (int, float)) or number <= 0:
            raise ValueError("Деление возможно только на положительное
число!")
        return Pair(round(self.first), round(self.second / number))

    # Перегрузка операторов сравнения
    def __eq__(self, other):
        if not isinstance(other, Pair):
            return False
        return self.cost() == other.cost()

    def __lt__(self, other):
        if not isinstance(other, Pair):
            raise TypeError("Сравнение возможно только между товарами!")
        return self.cost() < other.cost()

    def __le__(self, other):
        if not isinstance(other, Pair):
            raise TypeError("Сравнение возможно только между товарами!")
        return self.cost() <= other.cost()

    # Перегрузка оператора удаления
    def __del__(self):
        """
        Помимо удаления теперь и выводится информация об удаленном товаре.
        """
        print(f"Объект с ценой {self.first} и количеством {self.second}
удалён")


# Функция для создания объекта Pair
def make_pair(first, second):
    try:
        return Pair(first, second)
    except ValueError as e:
        print(f"Ошибка создания объекта: {e}")
        return None


# Пример использования
if __name__ == "__main__":
    # Создание двух объектов Pair
    print("Создание двух объектов Pair: ")
    pair1 = make_pair(20.0, 3)  # Цена: 20.0, Количество: 3
    pair2 = make_pair(15.0, 2)  # Цена: 15.0, Количество: 2

    if pair1 and pair2:
```

```
        print("\nДанные о первом товаре: ")
        pair1.display()
        print("Данные о втором товаре: ")
        pair2.display()

        # Сложение объектов
        print("\nТовар со средними характеристиками будет следующим: ")
        sum_pair = pair1 + pair2
        sum_pair.display()

        # Вычитание объектов
        print("\nВот что будет с первым товаром, если потратить все деньги со
вторго товара на первый товар:")
        sub_pair = pair1 - pair2
        sub_pair.display()

        # Умножение на число
        print("\nВот что будет, есть первого товара будет в 2 раза больше по
количеству: ")
        mul_pair = pair1 * 2
        mul_pair.display()

        # Деление на число
        print("\nВот что будет, есть первого товара будет в 2 раза меньше по
количеству: :")
        div_pair = pair1 / 2
        div_pair.display()

        # Сравнение объектов
        print("\nСравнение товаров:")
        if pair1 == pair2:
            print("\nОбщая стоимость первого товара совпадает с общей
стоимостью второго товара.\n")
        elif pair1 < pair2:
            print("\nОбщая стоимость первого товара меньще общей стоимость
второго товара.\n")
        else:
            print("\nОбщая стоимость первого товара больше общей стоимость
второго товара.\n")
```

В данном коде при помощи перегрузки операторов заданы свои, особые методы для сложения объектов, вычитания объектов, умножения и деления на число. Также добавлена перегрузка операторов сравнения для сравнения двух объектов одного класса. Также добавлена перегрузка оператора удаления объекта сборщиком мусора, теперь всякий раз, когда сборщик мусора будет удалять объект, он будет выводить информацию о нем.

Результаты работы программы с демонстрацией перегрузок всех приведенных ранее операторов (рис. 3).

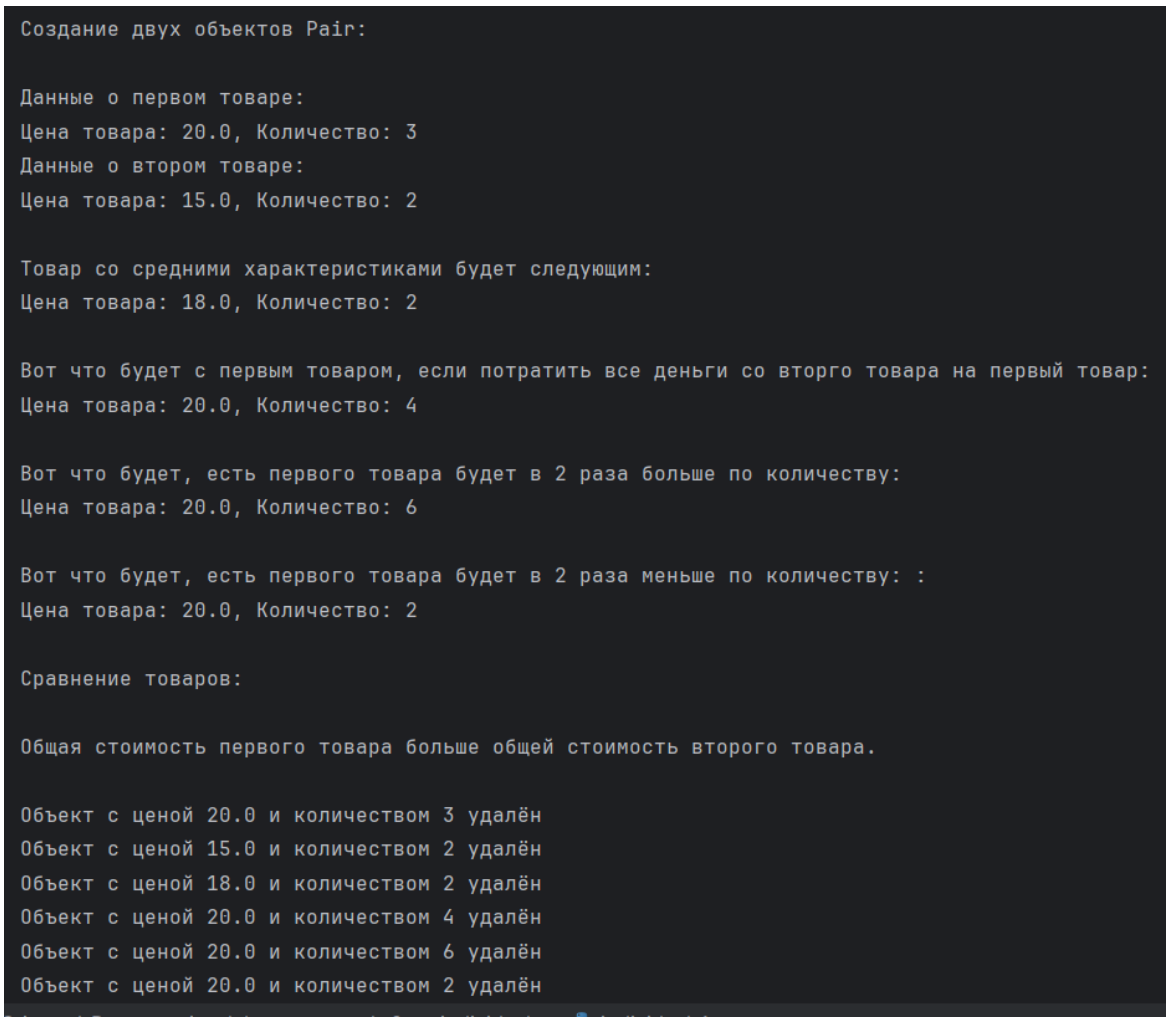

Рисунок 3 – Результаты работы программы индивидуального задания 1

Задание 2.

Необходимо дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле «size» должно храниться максимальное для данного объекта количество элементов списка; реализовать метод «size()», возвращающий установленную длину. Если количество элементов списка изменяется во время работы, определить в классе поле «count». Первоначальные значения «size» и «count» устанавливаются конструктором.

Создать класс Money для работы с денежными суммами. Сумма должна быть представлена списком, каждый элемент которого – десятичная цифра. Максимальная длина списка – 100 цифр, реальная длина задается

конструктором. Младший индекс соответствует младшей цифре денежной суммы. Младшие две цифры – копейки. (Вариант 25 (4)).

Код программы, выполняющей данные операции:

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

# Задание 2.
# Создать класс Money для работы с денежными суммами. Сумма должна быть
представлена списком,
# каждый элемент которого - десятичная цифра. Максимальная длина списка - 100
цифр, реальная
# длина задается конструктором. Младший индекс соответствует младшей цифре
денежной суммы.
# Младшие две цифры - копейки. (Вариант 25 (4)).

class Money:
    MAX_SIZE = 100  # Максимальная длина списка

    def __init__(self, amount):
        """
        Инициализация денежной суммы.
        :param amount: список цифр, где младший индекс соответствует младшей
цифре.
        """
        if len(amount) > Money.MAX_SIZE:
            raise ValueError(f"Длина списка не должна превышать
{Money.MAX_SIZE}.")

        # Проверяем, что все элементы списка - цифры от 0 до 9
        if not all(isinstance(d, int) and 0 <= d <= 9 for d in amount):
            raise ValueError("Все элементы списка должны быть цифрами от 0 до
9.")

        self.amount = amount[:]
        self._size = len(amount)  # Максимальная длина для объекта
        self._count = len(amount)  # Текущее количество элементов

    def size(self):
        """
        Возвращает максимальную длину для данного объекта.
        """
        return self._size

    def __len__(self):
        """
        Возвращает текущее количество элементов списка.
        """
        return self._count

    def __getitem__(self, index):
        """
        Позволяет обращаться к элементам суммы по индексу.
        :param index: индекс элемента.
        """
        if not (0 <= index < self._count):
            raise IndexError("Индекс выходит за пределы текущего количества
элементов.")
        return self.amount[index]
```

```
    def __setitem__(self, index, value):
        """
        Позволяет изменять элементы суммы по индексу.
        :param index: индекс элемента.
        :param value: новое значение (цифра от 0 до 9).
        """
        if not (0 <= index < self._count):
            raise IndexError("Индекс выходит за пределы текущего количества
элементов.")
        if not isinstance(value, int) or not (0 <= value <= 9):
            raise ValueError("Значение должно быть цифрой от 0 до 9.")
        self.amount[index] = value

    def append(self, digit):
        """
        Добавляет новую цифру в сумму.
        :param digit: цифра от 0 до 9.
        """
        if self._count >= Money.MAX_SIZE:
            raise ValueError("Достигнута максимальная длина списка для
данного объекта.")
        if not isinstance(digit, int) or not (0 <= digit <= 9):
            raise ValueError("Значение должно быть цифрой от 0 до 9.")
        self.amount.append(digit)
        self._count += 1

    def remove(self, index):
        """
        Удаляет элемент по индексу.
        :param index: индекс элемента.
        """
        if not (0 <= index < self._count):
            raise IndexError("Индекс выходит за пределы текущего количества
элементов.")
        del self.amount[index]
        self._count -= 1

    def __str__(self):
        """
        Возвращает строковое представление денежной суммы в формате рублей и
копеек.
        """
        if self._count < 2:
            return f"0.{''.join(map(str, self.amount[::-1]))} руб."

        rubles = ''.join(map(str, self.amount[2:][::-1])) or '0'
        kopecks = ''.join(map(str, self.amount[:2][::-1]))
        return f"{rubles}.{kopecks} руб."


# Пример использования
if __name__ == "__main__":
    money = Money([5, 0, 1])  # 105 рублей и 05 копеек
    print("Введенная сумма: ", money)  # Вывод: 1.05 руб.
    print("Размер:", money.size())  # Максимальная длина объекта
    print("Текущее количество элементов:", len(money))

    money.append(2)  # Добавляем цифру (увеличиваем сумму) и увеличиваем
размер списка
    print("Увеличенная сумма: ", money)  # Вывод: 21.05 руб.
    print("Размер:", money.size())  # Максимальная длина объекта
    print("Текущее количество элементов:", len(money))

    print("Младшая цифра: ", money[0])  # Доступ к младшей цифре: 5
```

```
    money[0] = 9  # Изменяем младшую цифру
    print("Изменение младшей цифры: ", money)

    money.remove(0)  # Удаляем младшую цифру
    print("Удаление младшей цифры: ", money)

    print("Размер:", money.size())  # Максимальная длина объекта
    print("Текущее количество элементов:", len(money))
```

Результаты работы программы (реализация работы с денежными суммами) (рис. 4).

```
Введенная сумма:  1.05 руб.
Размер: 3
Текущее количество элементов: 3
Увеличенная сумма:  21.05 руб.
Размер: 3
Текущее количество элементов: 4
Младшая цифра:  5
Изменение младшей цифры:  21.09 руб.
Удаление младшей цифры:  2.10 руб.
Размер: 3
Текущее количество элементов: 3
```

Рисунок 4 – Результаты работы программы индивидуального задания 2

Ссылка на репозиторий данной лабораторной работы:

https://github.com/AndreyPust/Object-Oriented_Programming_laboratory_work_2.git

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

В Python перегрузка операторов осуществляется с помощью методов, называемых «магическими методами» и чаще всего с двойными подчеркиваниями.

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

Для перегрузки арифметических операций используются следующие методы:

– __add__(self, other) — перегрузка оператора + (сложение)

– __sub__(self, other) — перегрузка оператора - (вычитание)

– __mul__(self, other) — перегрузка оператора * (умножение)

– __truediv__(self, other) — перегрузка оператора / (деление)

– __floordiv__(self, other) — перегрузка оператора // (целочисленное деление)

– __mod__(self, other) — перегрузка оператора % (остаток от деления)

– __pow__(self, other) — перегрузка оператора ** (возведение в степень)

– __and__(self, other) — перегрузка оператора & (логическое "И")

– __or__(self, other) — перегрузка оператора | (логическое "ИЛИ")

– __xor__(self, other) — перегрузка оператора ^ (логическое исключающее "ИЛИ")

Методы для перегрузки операций сравнения (отношения):

– __eq__(self, other) — перегрузка оператора ==

– __ne__(self, other) — перегрузка оператора !=

– __lt__(self, other) — перегрузка оператора <

– __le__(self, other) — перегрузка оператора <=

– __gt__(self, other) — перегрузка оператора >

– __ge__(self, other) — перегрузка оператора >=

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

__add__ (сложение): Этот метод вызывается, когда используется оператор + для объектов одного и того же типа (например, obj1 + obj2).

__iadd__ (сложение с присваиванием): Этот метод вызывается при использовании оператора +=. Он изменяет объект на месте, если это возможно.

__radd__ (реверсивное сложение): Этот метод вызывается, когда левый операнд не поддерживает сложение с правым операндом, и происходит

попытка обратного вызова. Например, other + obj, где other не поддерживает + с типом obj.

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

Метод __new__ отвечает за создание нового экземпляра класса и вызывается до __init__. Он используется для управления процессом создания объектов, особенно при наследовании от immutable (неизменяемых) типов, таких как int, str, tuple.

__new__ создаёт объект и возвращает его.

__init__ инициализирует уже созданный объект.

5. Чем отличаются методы __str__ и __repr__?

__str__ возвращает «человеко-читаемое» строковое представление объекта. Этот метод вызывается, когда используется функция str(), или когда объект выводится с помощью print(). Цель – сделать вывод более понятным для конечного пользователя.

__repr__ возвращает «машиночитаемое» строковое представление объекта. Этот метод должен возвращать строку, которая (по возможности) позволяет восстановить объект при использовании функции eval(). Если метод __str__ не определён, для вывода объекта вызывается __repr__.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python, также были изучены встроенные в Python функции называемые «магическими методами» и способы их перегрузки.