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

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

дисциплины «Объектно-ориентированное программирование»

Выполнил:
Горбунов Данила Евгеньевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А.-доцент департамента цифровых, роботехнических систем и электроники института перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой_____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Перегрузка операторов в языке Python.

Цель работы – приобретение навыков по перезагрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Выполнение работы

1. Выполнил пример 1, используя перегрузку операторов. (Рисунок 1)

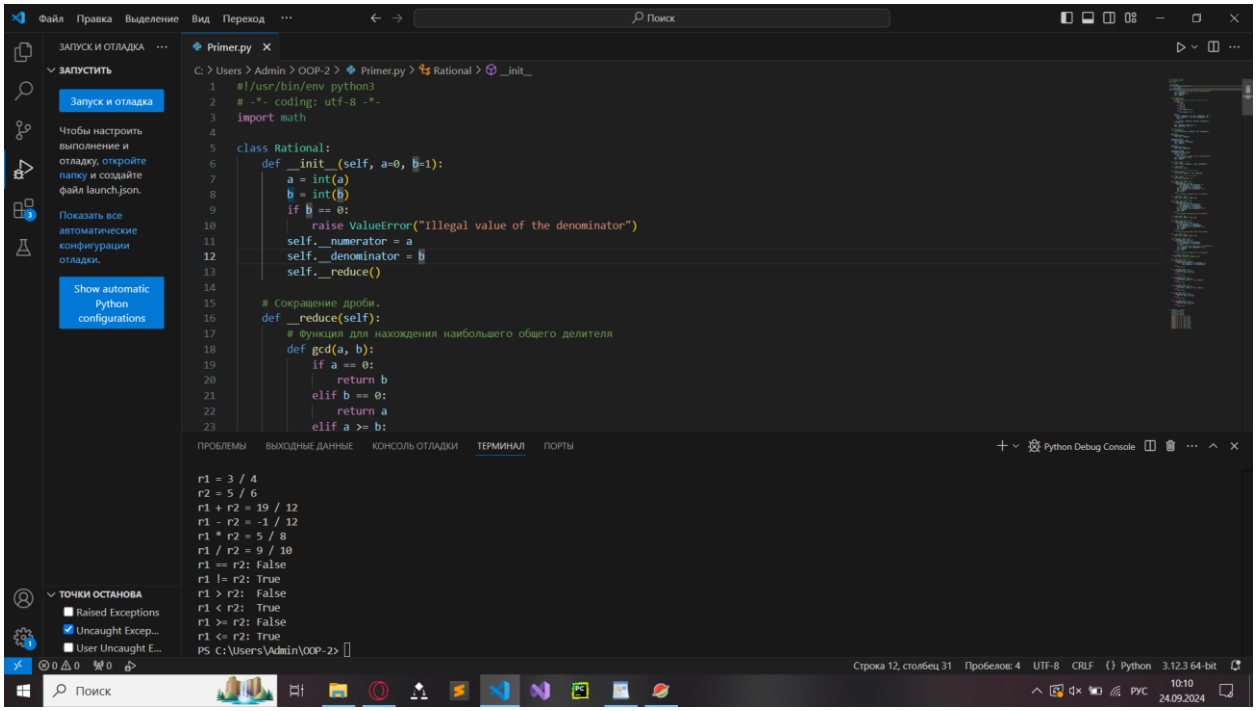

Рисунок 1. Пример 1

2. Выполнил индивидуальное задание 1 лабораторной работы 4.1, максимально задействовал имеющиеся в Python средства перегрузки операторов. (Рисунок 2)

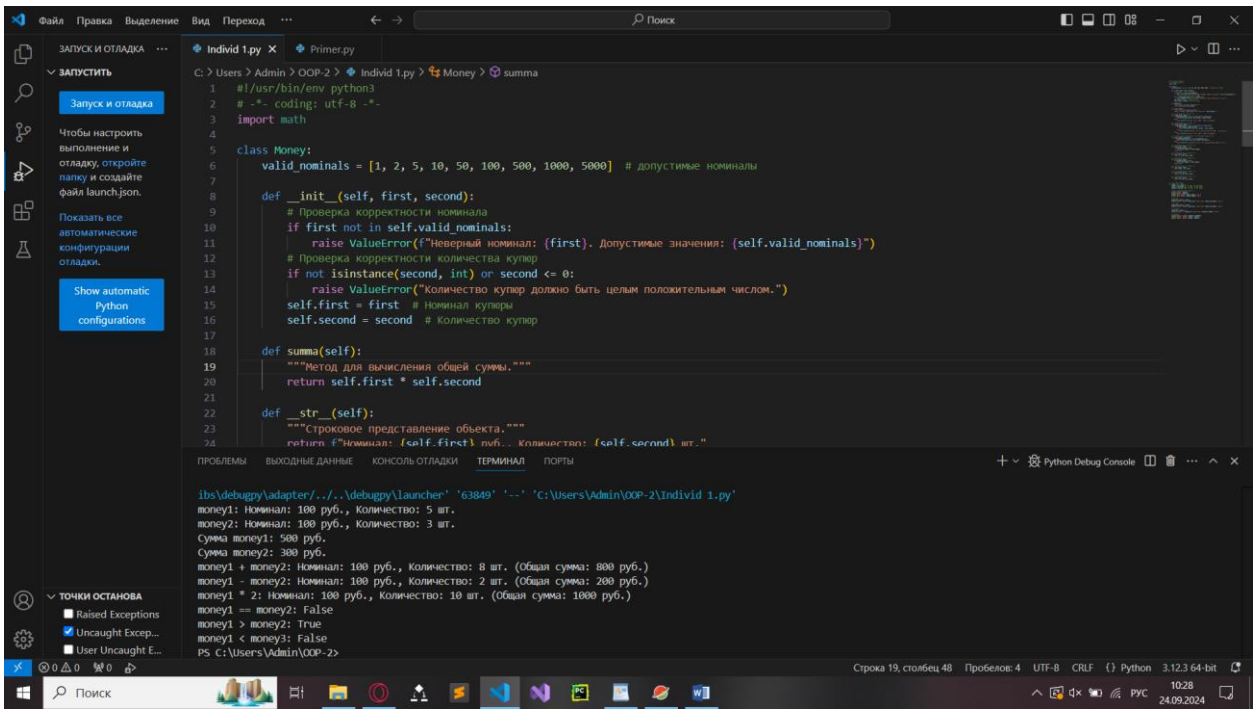

Рисунок 2. Индивидуальное задание 1

3. Выполнил индивидуальное задание 2, перегрузив операцию индексирования. (Рисунок 3)

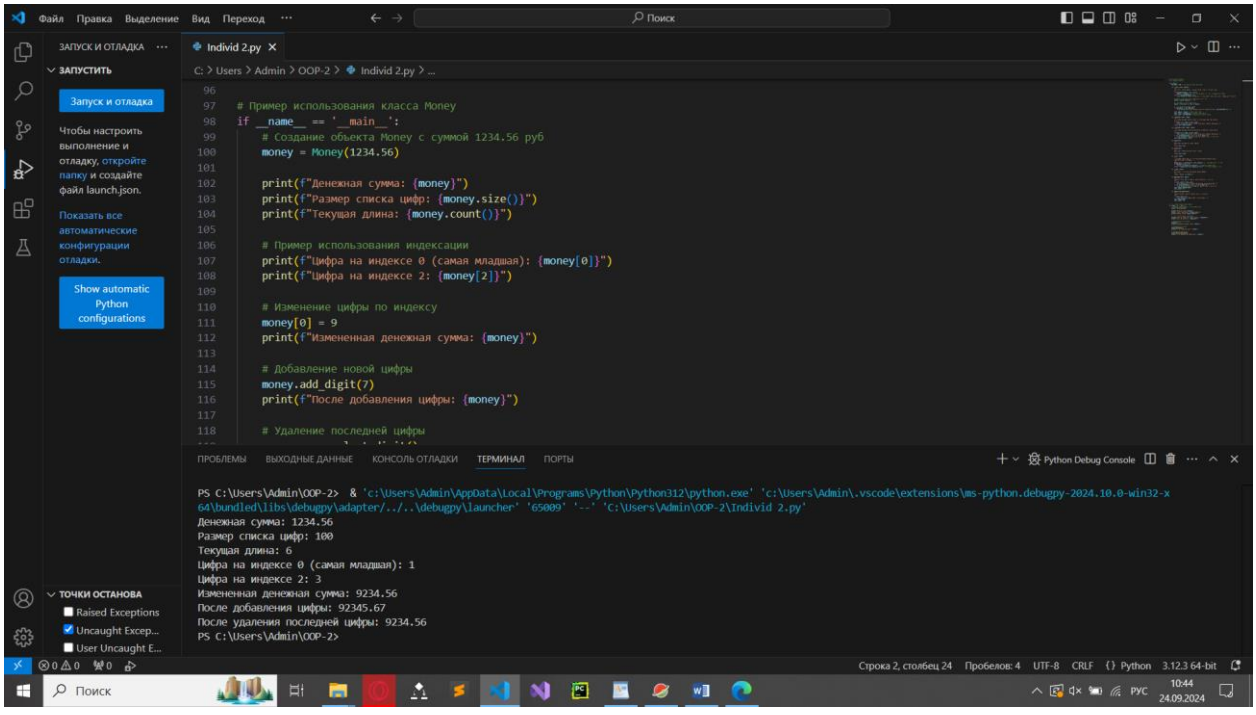

Рисунок 3. Индивидуальное задание 2

Контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

В Python для перегрузки операций используются специальные методы, называемые **магическими методами** (или **dunder methods** — от "double underscore"). Эти методы позволяют переопределить поведение операторов и других встроенных функций, таких как сложение, сравнение, доступ по индексу и т.д.

Примеры магических методов для перегрузки операций:

Арифметические операции: __add__, __sub__, __mul__, __truediv__ и др.

Операции сравнения: __eq__, __lt__, __le__, __ne__ и др.

Логические операции: __and__, __or__, __not__.

Операции над последовательностями: __getitem__, __setitem__, __len__.

Контекстные менеджеры: __enter__, __exit__.

Эти методы позволяют переопределить поведение операторов в классах и настраивать логику работы с пользовательскими объектами.

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

Магические методы для арифметических операций:

__add__(self, other) — перегрузка оператора сложения (+).

__sub__(self, other) — перегрузка оператора вычитания (-).

__mul__(self, other) — перегрузка оператора умножения (*).

__truediv__(self, other) — перегрузка оператора деления (/).

__floordiv__(self, other) — перегрузка оператора целочисленного деления (//).

__mod__(self, other) — перегрузка оператора остатка от деления (%).

__pow__(self, other) — перегрузка оператора возведения в степень (**).

Магические методы для операций отношения:

__eq__(self, other) — перегрузка оператора равенства (==).

__ne__(self, other) — перегрузка оператора неравенства (!=).

__lt__(self, other) — перегрузка оператора "меньше" (<).

__le__(self, other) — перегрузка оператора "меньше или равно" (<=).

__gt__(self, other) — перегрузка оператора "больше" (>).

__ge__(self, other) — перегрузка оператора "больше или равно" (>=).

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

__add__: Этот метод вызывается, когда используется оператор сложения (+). Он отвечает за обычное сложение двух объектов.

```
class Number:
    def __init__(self, value):
        self.value = value
    def __add__(self, other):
        return Number(self.value + other.value)
n1 = Number(3)
n2 = Number(5)
```

```
n3 = n1 + n2  # Вызовет __add__
print(n3.value)  # Output: 8
```

__iadd__: Этот метод вызывается, когда используется оператор добавления с присваиванием (+=). Он модифицирует объект на месте, если это возможно.

```
class Number:
    def __init__(self, value):
        self.value = value
    def __iadd__(self, other):
        self.value += other.value
        return self
n1 = Number(3)
n2 = Number(5)
n1 += n2  # Вызовет __iadd__
print(n1.value)  # Output: 8
```

__radd__: Этот метод вызывается, если объект слева не поддерживает сложение с объектом справа, и вызывается перегрузка для правого операнда. То есть если первый объект не реализует __add__, Python попробует вызвать __radd__ у второго объекта.

```
class Number:
    def __init__(self, value):
        self.value = value
    def __radd__(self, other):
        return Number(self.value + other)
n1 = Number(3)
result = 5 + n1  # Вызовет __radd__, т.к. int не знает как складывать с Number
print(result.value)  # Output: 8
```

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

__new__ — это метод, который отвечает за создание нового экземпляра класса. Он вызывается до __init__ и именно он отвечает за выделение памяти для нового объекта. __new__ возвращает новый экземпляр класса.

__init__ — это метод, который отвечает за инициализацию объекта. После создания объекта с помощью __new__, метод __init__ инициализирует его атрибуты. Он не создаёт объект, а работает с уже существующим.

Пример, когда нужно использовать __new__, — если класс наследуется от неизменяемого типа, такого как int, str, или для реализации шаблона проектирования Singleton.

```
class Singleton:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
    def __init__(self):
        print("Singleton created")
s1 = Singleton()
s2 = Singleton()
print(s1 is s2)  # Output: True
```

5. Чем отличаются методы __str__ и __repr__?

__str__ предназначен для создания удобочитаемого представления объекта для пользователя. Его основная цель — отображение объекта в «человеческом» формате, когда вызывается функция print() или str(). Результат должен быть понятным для конечного пользователя.

__repr__ предназначен для создания строкового представления объекта, которое однозначно его описывает, часто с целью использования в отладке. __repr__ должен возвращать строку, которая может быть использована для создания аналогичного объекта (если это возможно).

Если не реализован метод __str__, то вызывается __repr__.

```
class Example:
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return f"Example with value {self.value}"
    def __repr__(self):
        return f"Example({self.value})"
ex = Example(42)
print(str(ex))   # Example with value 42 (читаемое представление)
print(repr(ex))  # Example(42) (представление для отладки)
```

Вывод: в ходе данной работы приобрёл навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.