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

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
дисциплины «Объектно-ориентированное программирование»

Выполнила:
Гайчук Дарья Дмитриевна
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 перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создала новый репозиторий и клонировала его на свой компьютер.

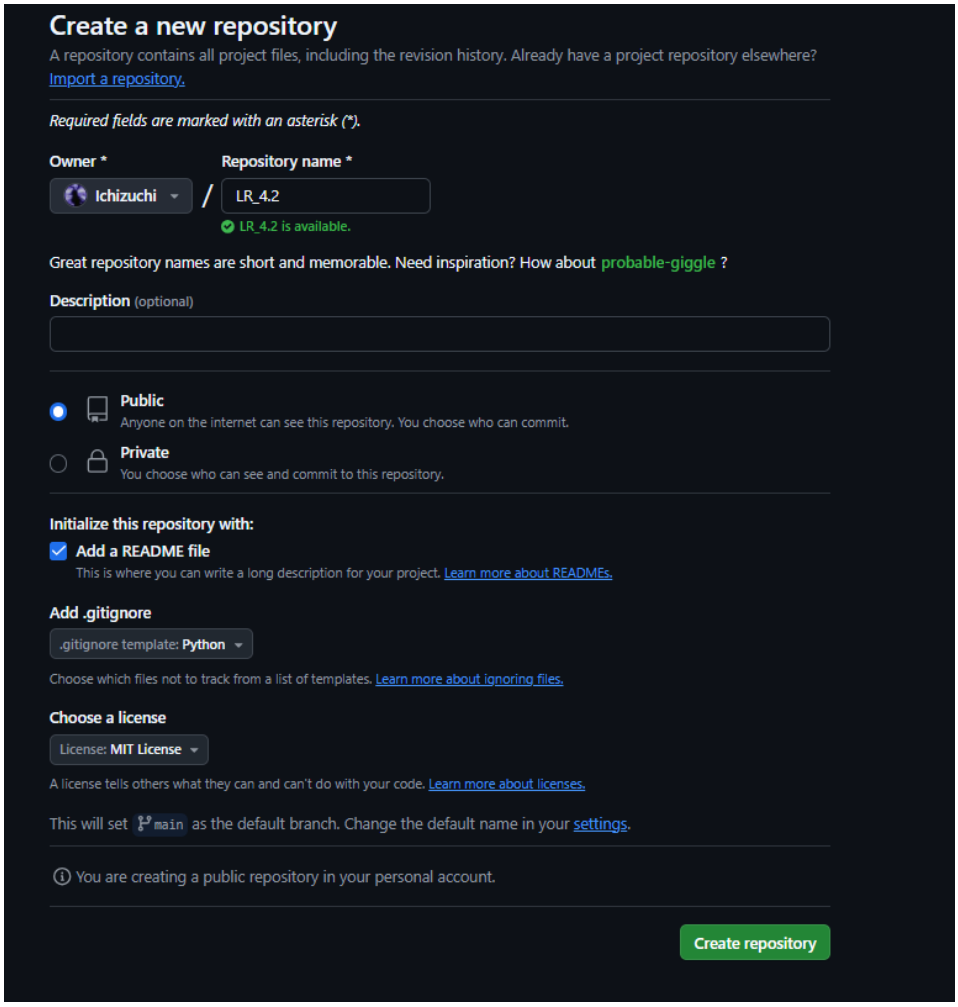


Рисунок 1. Создание репозитория

2. Клонировала репозиторий на свой компьютер.

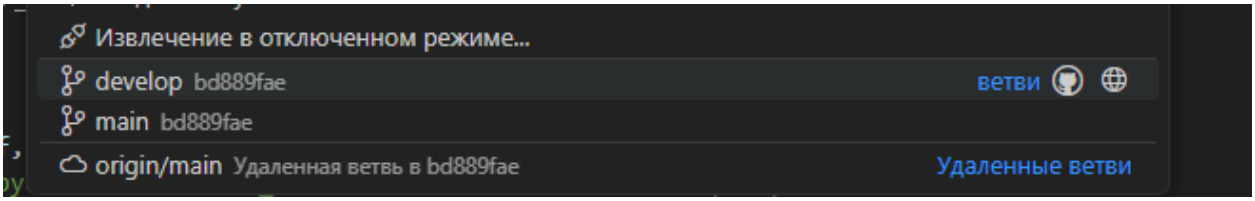


Рисунок 2. Модель ветвления git-flow

3. Создала виртуальное окружение Anaconda с именем репозитория.

```
@Ichizuchi ➜/workspaces/LR_4.2 (main) $ conda create -n myenv python=3.10
Retrieving notices: ...working... done
Channels:
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/conda/envs/myenv

  added / updated specs:
    - python=3.10


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
```

Рисунок 3. Создание виртуального окружения

Выполнение примеров.

Изменить класс Rational из примера 1 лабораторной работы 4.1, используя перегрузку операторов.

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

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

ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ  КОММЕНТАРИИ

/home/codespace/.python/current/bin/python3 /workspaces/LR_4.2/Examples/Ex_1.py
@Ichizuchi ➜/workspaces/LR_4.2 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.2/Examples/Ex_1.py
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
@Ichizuchi ➜/workspaces/LR_4.2 (develop) $
```

Рисунок 4. Выполнение примера

Выполнение индивидуального задания.

Вариант 3

Задание №1.

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

```python
    def __sub__(self, other):
        elif isinstance(other, int):
            return Pair(self.first - other, self.second - other)
        else:
            raise TypeError("Операнд должен быть либо объектом Pair, либо целым числом.")

    # Перегрузка оператора умножения *
    def __mul__(self, other):
        if isinstance(other, Pair):
            return Pair(self.first * other.first, self.second * other.second)
        elif isinstance(other, int):
            return Pair(self.first * other, self.second * other)
        else:
            raise TypeError("Операнд должен быть либо объектом Pair, либо целым числом.")

    # Перегрузка оператора деления /
    def __truediv__(self, other):
        if isinstance(other, Pair):
            if other.first == 0 or other.second == 0:
                raise ZeroDivisionError("Деление на ноль невозможно.")
            return Pair(self.first // other.first, self.second // other.second)
        elif isinstance(other, int):
            if other == 0:
                raise ZeroDivisionError("Деление на ноль невозможно.")
            return Pair(self.first // other, self.second // other)
        else:
            raise TypeError("Операнд должен быть либо объектом Pair, либо целым числом.")

    # Перегрузка оператора равенства ==
    def __eq__(self, other):
        if isinstance(other, Pair):
            return self.first == other.first and self.second == other.second
        return False

    # Перегрузка оператора неравенства !=
    def __ne__(self, other):
        return not self.__eq__(other)


def make_pair(first, second):
    try:
        return Pair(first, second)
    except ValueError as e:
        print(f"Ошибка: {e}")
        exit(1)
```

Рисунок 5. Код программы

```python
    pair1.display()
    print(f"Целая часть дроби: {pair1.ipart()}")

    pair2.display()
    print(f"Целая часть дроби: {pair2.ipart()}")

    print("\nОперации с перегруженными операторами:")
    print(f"pair1 + pair2: {pair1 + pair2}")
```

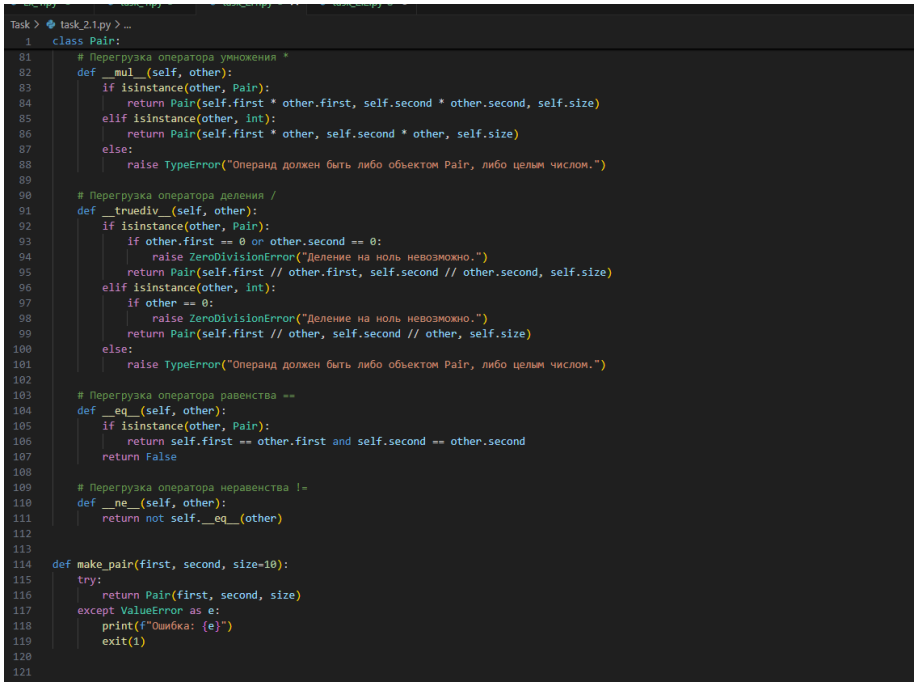
```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ  КОММЕНТАРИИ

Деление Hex1 на Hex2:
0x2
@Ichizuchi ➜/workspaces/LR_4.2 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.2/Task/task_1.py
first = 5, second = 2
Целая часть дроби: 2
first = 3, second = 1
Целая часть дроби: 3

Операции с перегруженными операторами:
pair1 + pair2: <__main__.Pair object at 0x7d2a8e6ab710>
pair1 - pair2: <__main__.Pair object at 0x7d2a8e6ab710>
pair1 * pair2: <__main__.Pair object at 0x7d2a8e6ab710>
pair1 / pair2: <__main__.Pair object at 0x7d2a8e6ab710>
pair1 == pair2: False
pair1 != pair2: True
Введите целое положительное число first: 6
Введите целое положительное число second: 3
first = 6, second = 3
Целая часть дроби: 2
@Ichizuchi ➜/workspaces/LR_4.2 (develop) $
```

Рисунок 6. Результат выполнения

Задание №2.

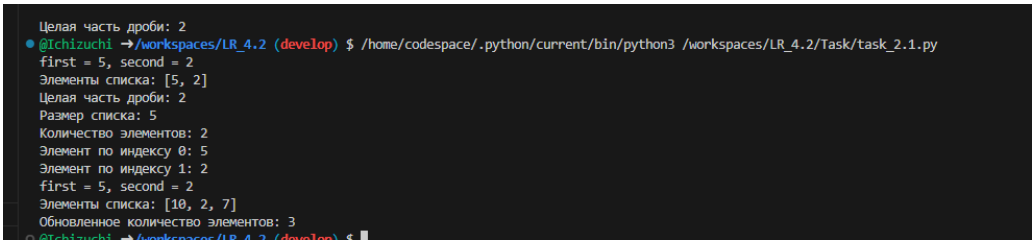
Дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования [ ]. Максимально возможный размер списка задать константой. В отдельном поле size должно храниться максимальное для

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.



```
Task > task_2.1.py > ...
class Pair:
    # Перегрузка оператора умножения *
    def __mul__(self, other):
        if isinstance(other, Pair):
            return Pair(self.first * other.first, self.second * other.second, self.size)
        elif isinstance(other, int):
            return Pair(self.first * other, self.second * other, self.size)
        else:
            raise TypeError("Операнд должен быть либо объектом Pair, либо целым числом.")

    # Перегрузка оператора деления /
    def __truediv__(self, other):
        if isinstance(other, Pair):
            if other.first == 0 or other.second == 0:
                raise ZeroDivisionError("Деление на ноль невозможно.")
            return Pair(self.first // other.first, self.second // other.second, self.size)
        elif isinstance(other, int):
            if other == 0:
                raise ZeroDivisionError("Деление на ноль невозможно.")
            return Pair(self.first // other, self.second // other, self.size)
        else:
            raise TypeError("Операнд должен быть либо объектом Pair, либо целым числом.")

    # Перегрузка оператора равенства ==
    def __eq__(self, other):
        if isinstance(other, Pair):
            return self.first == other.first and self.second == other.second
        return False

    # Перегрузка оператора неравенства !=
    def __ne__(self, other):
        return not self.__eq__(other)


def make_pair(first, second, size=10):
    try:
        return Pair(first, second, size)
    except ValueError as e:
        print(f"Ошибка: {e}")
        exit(1)
```

Рисунок 7. Код программы



```
Целая часть дроби: 2
@Ichizuchi ➜/workspaces/LR_4.2 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.2/Task/task_2.1.py
first = 5, second = 2
Элементы списка: [5, 2]
Целая часть дроби: 2
Размер списка: 5
Количество элементов: 2
Элемент по индексу 0: 5
Элемент по индексу 1: 2
first = 5, second = 2
Элементы списка: [10, 2, 7]
Обновленное количество элементов: 3
```

Рисунок 8. Результат выполнения

В тех задачах, где возможно, реализовать конструктор инициализации строкой создать класс Hex для работы с беззнаковыми целыми шестнадцатеричными числами, используя для представления числа список из 100 элементов типа int, каждый из которых является шестнадцатеричной цифрой. Младшая цифра имеет меньший индекс. Реальный размер списка задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых и операции сравнения.

```python
class Hex:
    MAX_SIZE = 100

    def __init__(self, hex_string='0'):
        # Инициализируем список из MAX_SIZE элементов, каждый из которых равен 0
        self.hex_digits = [0] * self.MAX_SIZE
        hex_string = hex_string.lstrip('0x').upper()
        real_size = min(len(hex_string), self.MAX_SIZE)

        for i in range(real_size):
            digit = hex_string[-(i + 1)]
            self.hex_digits[i] = int(digit, 16)

        self.size = real_size

    def display(self):
        # Выводим число в виде строки (начиная с самого старшего разряда)
        hex_str = ''.join(hex(digit)[2:].upper() for digit in reversed(self.hex_digits[:self.size]))
        print(f"0x{hex_str}")

    # Перегрузка оператора сложения +
    def __add__(self, other):
        if not isinstance(other, Hex):
            raise TypeError("Операнд должен быть объектом класса Hex.")

        result = Hex()  # Создаем новый объект для результата
        carry = 0

        # Складываем побитно числа
        for i in range(self.MAX_SIZE):
            total = self.hex_digits[i] + other.hex_digits[i] + carry
            result.hex_digits[i] = total % 16  # Остаток от деления на 16
            carry = total // 16  # Перенос в старший разряд

        result.size = max(self.size, other.size) + (1 if carry else 0)
        return result

    def __sub__(self, other):
        if not isinstance(other, Hex):
            raise TypeError("Операнд должен быть объектом класса Hex.")

        result = Hex()
        borrow = 0

        for i in range(self.MAX_SIZE):
            total = self.hex_digits[i] - other.hex_digits[i] - borrow
            if total < 0:
                total += 16
```

Рисунок 9. Код программы

```
0x2
@Ichizuchi →/workspaces/LR_4.2 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_4.2/Task/task_2.2.py
Hex1:
0x1A3
Hex2:
0xB2
Сумма Hex1 и Hex2:
0x255
Разность Hex1 и Hex2:
0x0F1
Произведение Hex1 и Hex2:
0x0C156
Деление Hex1 на Hex2:
0x2
@Ichizuchi →/workspaces/LR_4.2 (develop) $
```

Рисунок 10. Результат выполнения

```
@Ichizuchi →/workspaces/LR_4.2 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
@Ichizuchi →/workspaces/LR_4.2 (main) $ git merge develop
Updating bd889fa..2dc35cf
Fast-forward
 Examples/Ex_1.py | 175 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Task/task_1.py   | 109 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Task/task_2.1.py | 139 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Task/task_2.2.py | 133 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 556 insertions(+)
 create mode 100644 Examples/Ex_1.py
 create mode 100644 Task/task_1.py
 create mode 100644 Task/task_2.1.py
 create mode 100644 Task/task_2.2.py
@Ichizuchi →/workspaces/LR_4.2 (main) $ conda env export > environment.yml
@Ichizuchi →/workspaces/LR_4.2 (main) $ pip freeze > requirements.txt
@Ichizuchi →/workspaces/LR_4.2 (main) $ conda init
no change     /opt/conda/condabin/conda
```

Рисунок 11. Слияние веток

Ответы на 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 стандартных операторов для объектов пользовательских классов. Например:

Арифметические операции («+», «-», «*», «/» и др.) — методы вроде «__add__», «__sub__», «__mul__», «__truediv__».

Операции сравнения («<«, «>«, «==«, «!=«, «<=«, «>=«) — методы вроде «__lt__», «__gt__», «__eq__», «__ne__».

Другие операторы — можно перегружать операторы индексирования, приведения типов, вызова функций, и т.д., например, с помощью методов «__getitem__», «__setitem__», «__call__», «__int__», «__float__», и других.

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

Для арифметических операций существуют следующие магические методы:

- «__add__(self, other)» — для перегрузки оператора сложения «+».
- «__sub__(self, other)» — для вычитания «-».
- «__mul__(self, other)» — для умножения «*».
- «__truediv__(self, other)» — для деления «/».
- «__floordiv__(self, other)» — для целочисленного деления «//».
- «__mod__(self, other)» — для нахождения остатка «%».
- «__pow__(self, other)» — для возведения в степень «**».

Для операций сравнения:

- «__lt__(self, other)» — для операции "меньше" «<«.
- «__le__(self, other)» — для "меньше или равно" «<=«.
- «__eq__(self, other)» — для "равно" «==«.
- «__ne__(self, other)» — для "не равно" «!=«.
- «__gt__(self, other)» — для "больше" «>«.
- «__ge__(self, other)» — для "больше или равно" «>=«.

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

«__add__(self, other)» — вызывается при использовании оператора сложения «+» для объекта класса:

```
class A:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return A(self.value + other.value)
a1 = A(5)
a2 = A(3)
a3 = a1 + a2  # Вызовет метод __add__
print(a3.value)  # 8
```

«__iadd__(self, other)» — вызывается для перегрузки операции "сложение с присваиванием" («+=«):

```
class A:
    def __init__(self, value):
        self.value = value

    def __iadd__(self, other):
        self.value += other.value
        return self
a1 = A(5)
a2 = A(3)
a1 += a2  # Вызовет метод __iadd__
print(a1.value)  # 8
```

«__radd__(self, other)» — вызывается, если левый операнд не поддерживает операцию сложения, и тогда операция передается правому операнду.

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

Метод «__new__» используется для создания нового экземпляра класса. Он вызывается перед «__init__» и возвращает сам объект. Основное его предназначение — контроль за процессом создания объекта, например, в случае использования паттерна одиночка (Singleton) или при наследовании от неизменяемых типов данных, таких как «int», «str», «tuple».

```
class Singleton:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Singleton, cls).__new__(cls)
        return cls._instance
    def __init__(self):
        self.value = 42
s1 = Singleton()
s2 = Singleton()
print(s1 is s2)  # True
```

«__init__» же вызывается после создания объекта и занимается его инициализацией (заполнение полей, начальная настройка).

Основные отличия:

«__new__» создает новый объект, возвращая его.

«__init__» инициализирует уже существующий объект и ничего не возвращает.

5. Чем отличаются методы «__str__» и «__repr__»?

- «__str__» — возвращает строковое представление объекта для пользователя, которое должно быть максимально понятным и удобным для чтения. Этот метод используется при вызове «print()» и «str()».

- «__repr__» — возвращает строковое представление объекта, которое должно быть однозначным и технически точным. Используется для отладки и

отображает объект так, чтобы его можно было воссоздать с помощью «eval()». Этот метод используется в интерактивной оболочке и при вызове «repr()».

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