

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ**

ФЕДЕРАЦИИ

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

«Перегрузка операторов в языке Python»

Отчет по лабораторной работе № 4.2

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Кучеренко С. Ю. _____ « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы.

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

import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Vector2D({}, {})'.format(self.x, self.y)

    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y
        return self

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __isub__(self, other):
        self.x -= other.x
        self.y -= other.y
        return self

    def __abs__(self):
```

```

        return math.hypot(self.x, self.y)

    def __bool__(self):
        return self.x != 0 or self.y != 0

    def __neg__(self):
        return Vector2D(-self.x, -self.y)

if __name__ == '__main__':
    x = Vector2D(3, 4)
    print(x)
    print(abs(x))
    y = Vector2D(5, 6)
    print(y)
    print(x + y)
    print(x - y)
    print(-x)
    x += y
    print(x)
    print(bool(x))
    z = Vector2D(0, 0)
    print(bool(z))
    print(-z)

```

```

(3, 4)
5.0
(5, 6)
(8, 10)
(-2, -2)
(-3, -4)
(8, 10)
True
False
(0, 0)

```

Рисунок 1 – Результат выполнения программы

```

#!/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}")

```

```
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
```

Рисунок 2 – Результат выполнения программы

8. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

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

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

Поле first – целое положительное число, часы; поле second – целое
положительное
число, минуты. Реализовать метод minutes() – приведение времени в минуты.
"""

class Conversion:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes

    # строковое представление времени
    def __str__(self):
        return f"{self.hours:02d}:{self.minutes:02d}"

    # сумма времен
    def __add__(self, other):
        total_minutes = self.minutes + other.minutes
        carry_hours = total_minutes // 60
        total_minutes %= 60
```

```

        total_hours = self.hours + other.hours + carry_hours
        return Conversion(total_hours, total_minutes)

    # разность времен
    def __sub__(self, other):
        total_minutes = (self.hours * 60 + self.minutes) - (other.hours * 60 +
other.minutes)
        if total_minutes < 0:
            raise ValueError("Результат отрицательный")
        total_hours = total_minutes // 60
        total_minutes %= 60
        return Conversion(total_hours, total_minutes)

    # сравнение времен
    def __lt__(self, other):
        return self.minutes + self.hours * 60 < other.minutes + other.hours *
60

    # время в минутах
    def get_minutes(self):
        return self.hours * 60 + self.minutes

if __name__ == "__main__":
    time1 = Conversion(2, 30)
    time2 = Conversion(1, 45)

    print(time1)
    print(time2)

    sum_time = time1 + time2
    print(sum_time)

    diff_time = time1 - time2
    print(diff_time)

    print(time1 < time2)

    print(time1.get_minutes())
    print(time2.get_minutes())

```

```

02:30
01:45
04:15
00:45
False
150
105

Process finished with exit code 0

```

Рисунок 3 – Результат выполнения программы


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

"""
Товарный чек содержит список товаров, купленных покупателем в магазине. Один
элемент
списка представляет собой пару: товар-сумма. Товар — это класс Goods с полями
кода и
наименования товара, цены за единицу товара, количества покупаемых единиц
товара. В
классе должны быть реализованы методы доступа к полям для получения и
изменения
информации, а также метод вычисления суммы оплаты за товар. Для моделирования
товарного чека реализовать класс Receipt, полями которого являются номер
товарного чека,
дата и время его создания, список покупаемых товаров. В классе Receipt
реализовать
методы добавления, изменения и удаления записи о покупаемом товаре, метод
поиска
информации об определенном виде товара по его коду, а также метод подсчета
общей
суммы, на которую были осуществлены покупки. Методы добавления и изменения
принимают в качестве аргумента объект класса Goods. Метод поиска возвращает
объект
класса Goods в качестве результата.
"""

class Goods:
    def __init__(self, code, name, price, quantity):
        self.code = code
        self.name = name
        self.price = price
        self.quantity = quantity

    # Возвращаем код товара
    def get_code(self):
        return self.code

    # Устанавливаем новый код товара
    def set_code(self, code):
        self.code = code

    # Возвращаем название товара
    def get_name(self):
        return self.name

    # Новое название товара
    def set_name(self, name):
        self.name = name

    # Цена за ед. товара
    def get_price(self):
        return self.price

    # Новая цена за ед. товара
    def set_price(self, price):
        self.price = price

    # Количество товара, который купили
    def get_quantity(self):
        return self.quantity

    # Новое количество товара, который купили
```

```

def set_quantity(self, quantity):
    self.quantity = quantity

# Общая стоимость покупки
def calculate_total_price(self):
    return self.price * self.quantity

class Receipt:
    def __init__(self, receipt_number, date_time):
        self.receipt_number = receipt_number
        self.date_time = date_time
        self.items = []

    # Добавляем покупку в чек
    def add_item(self, goods):
        self.items.append(goods)

    def update_item(self, code, new_goods):
        for i in range(len(self.items)):
            if self.items[i].get_code() == code:
                self.items[i] = new_goods
                return True
        return False

    # Удаляем товар из чека по коду
    def remove_item(self, code):
        for i in range(len(self.items)):
            if self.items[i].get_code() == code:
                del self.items[i]
                return True
        return False

    # Поиск товара по коду
    def find_item_by_code(self, code):
        for item in self.items:
            if item.get_code() == code:
                return item
        return None

    # Общая сумма покупок
    def calculate_total_amount(self):
        total_amount = 0
        for item in self.items:
            total_amount += item.calculate_total_price()
        return total_amount

if __name__ == "__main__":

    item1 = Goods("001", "Хлеб", 10, 2)
    item2 = Goods("002", "Молоко", 5, 3)

    receipt = Receipt("0001", "20.05.2023 22:00")
    print("Номер чека:", receipt.receipt_number)
    print("Дата и время создания:", receipt.date_time)

    # Добавление товаров в чек
    receipt.add_item(item1)
    receipt.add_item(item2)

    # Изменение количества товара в чеке
    updated_item = Goods("001", "Хлеб", 10, 5)
    receipt.update_item("001", updated_item)

    # Удаление товара из чека

```

```
receipt.remove_item("001")

# Поиск информации о товаре по его коду
found_item = receipt.find_item_by_code("002")
if found_item:
    print("Найден товар:", found_item.get_name())
else:
    print("Товар не найден.")

# Подсчет общей суммы покупок
total_amount = receipt.calculate_total_amount()
print("Общая сумма покупок:", total_amount)
```

```
Номер чека: 0001
Дата и время создания: 20.05.2023 22:00
Найден товар: Молоко
Общая сумма покупок: 15

Process finished with exit code 0
```

Рисунок 4 – Результат выполнения программы

9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой main / master.
11. Отправьте сделанные изменения на сервер GitHub.

Контрольные 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.

Например, у нас есть два класса:

```
class A:
    def go(self):
        print('Go, A!')

class B(A):
    def go(self, name):
        print('Go, {}!'.format(name))
```

В данном примере класс *B* наследует класс *A*, но переопределяет метод *go*, поэтому он имеет мало общего с аналогичным методом класса *A*.

Однако в python имеются методы, которые, как правило, не вызываются напрямую, а вызываются встроенными функциями или операторами.

Например, метод `__init__` перегружает конструктор класса. Конструктор - создание экземпляра класса.

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 $(x \wedge y)$.
- `__or__(self, other)` - битовое ИЛИ $(x | y)$.

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

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