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»
Вариант 15

Выполнил:
Степанов Леонид Викторович
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)

Проверил:
Богданов Сергей Сергеевич,
Ассистент департамента цифровых, робототехнических систем и электроники

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Элементы объектно-ориентированного программирования в языке Python

Цель: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Общее задание:

В лабораторной работе 1 был изменен пример так, чтобы к нему были добавлены перегрузки способа реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе, в частности на рис. 1 отображён результат выполнения кода, где реализованы математические и логические операции. На рисунке 1 отображён результат выполнения кода:

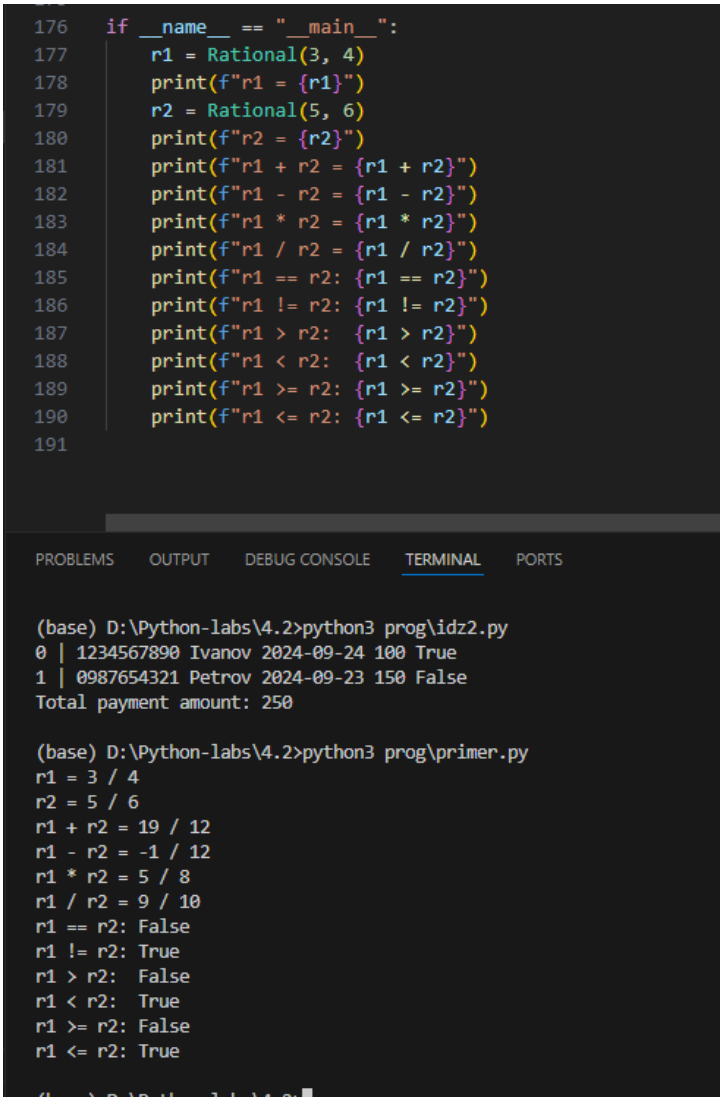
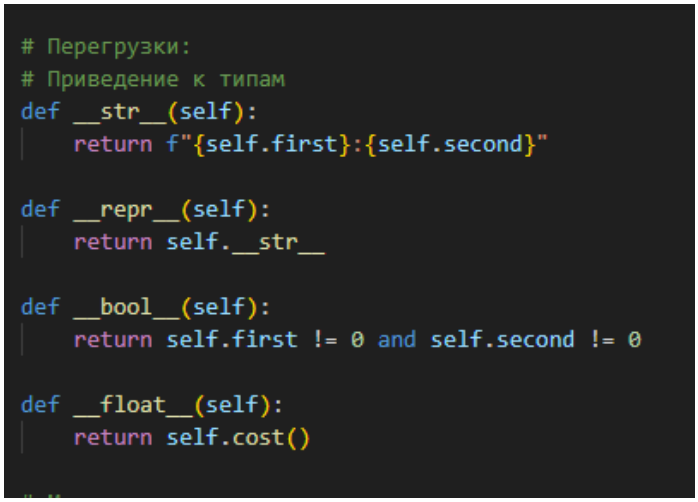


Рисунок 1 – Результат выполнения

Индивидуальное задание (Вариант 15)

Задание 1: Необходимо к прошлому индивидуальному заданию добавить перегрузки.

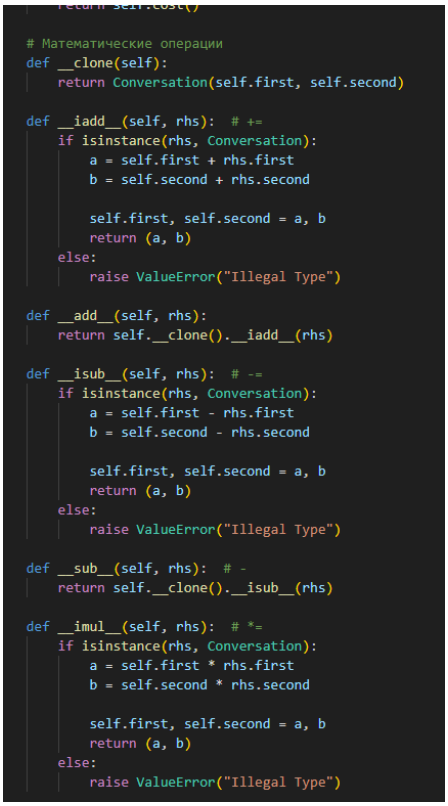
Перегрузки необходимые для приведения к различным типам отображены на рис. 2



```
# Перегрузки:
# Приведение к типам
def __str__(self):
    return f"{self.first}:{self.second}"

def __repr__(self):
    return self.__str__

def __bool__(self):
    return self.first != 0 and self.second != 0

def __float__(self):
    return self.cost()
```

Рисунок 2 Приведение к типам

Перегрузки необходимые для выполнения математических операций отображены на рис. 3



```
# Математические операции
def __clone(self):
    return Conversation(self.first, self.second)

def __iadd__(self, rhs):  # +=
    if isinstance(rhs, Conversation):
        a = self.first + rhs.first
        b = self.second + rhs.second

        self.first, self.second = a, b
        return (a, b)
    else:
        raise ValueError("Illegal Type")

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

def __isub__(self, rhs):  # -=
    if isinstance(rhs, Conversation):
        a = self.first - rhs.first
        b = self.second - rhs.second

        self.first, self.second = a, b
        return (a, b)
    else:
        raise ValueError("Illegal Type")

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

def __imul__(self, rhs):  # *=
    if isinstance(rhs, Conversation):
        a = self.first * rhs.first
        b = self.second * rhs.second

        self.first, self.second = a, b
        return (a, b)
    else:
        raise ValueError("Illegal Type")
```

Рисунок 3 – Математические операции

Перегрузки необходимые для выполнения логических операций отображены на рис. 4

```python
def __eq__(self, rhs):
    """Перегрузка оператора равенства"""
    if isinstance(rhs, Conversation):
        return self.cost() == rhs.cost()

    return NotImplemented

def __lt__(self, rhs):
    """Перегрузка оператора меньше"""
    if isinstance(rhs, Conversation):
        return self.cost() < rhs.cost()
    return NotImplemented
```

Рисунок 4 – Логические операции

Пример работы программы на рис. 5

```python
if __name__ == "__main__":
    conv1 = make_Conversation(12, 3.5)
    conv2 = make_Conversation(10, 2.5)

    if conv1 and conv2:
        conv1.display()
        conv2.display()

        # Сравнение объектов
        if conv1 == conv2:
            print("Разговоры имеют одинаковую ст
        elif conv1 < conv2:
            print("Первый разговор дешевле второ
        else:
            print("Первый разговор дороже второг

        print(float(conv1))
        print(conv1 - conv2)
        print(conv1 + conv2)
        print(conv1 * conv2)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(base) D:\Python-labs\4.2>python3 prog\idz1.py
Продолжительность разговора: 12 минут
Стоимость одной минуты: 3.5 рубля
Общая стоимость разговора: 42.0 рублей
Продолжительность разговора: 10 минут
Стоимость одной минуты: 2.5 рубля
Общая стоимость разговора: 25.0 рублей
Первый разговор дороже второго.
42.0
(2, 1.0)
(22, 6.0)
(120, 8.75)

(base) D:\Python-labs\4.2>
```

Рисунок 5 – Результат работы

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.

Используя класс Bill, реализовать класс ListPayer. Класс содержит список плательщиков за телефонные услуги, дату создания списка, номер списка. Поля одного элемента списка — это: плательщик (класс Bill), признак оплаты, дата платежа, сумма платежа. Реализовать методы добавления плательщиков в список и удаления их из него; метод поиска плательщика по номеру телефона и по фамилии, по дате платежа. Метод вычисления полной стоимости платежей всего списка. Реализовать операцию объединения и операцию пересечения списков. Реализовать операцию генерации конкретного объекта Group (группа), содержащего список плательщиков, из объекта типа ListPayer. Должна быть возможность выбирать группу плательщиков по признаку оплаты, по атрибутам, по дате платежа, по номеру телефона.

Реализация класса Bill (рис. 6) содержит в себе атрибуты: номер телефона, фамилия, дата оплаты, сумма оплаты, оплачено ли.

```
class Bill:
    def __init__(self, phone_number, last_name, payment_date, payment_amount, paid):
        self.phone_number = phone_number
        self.last_name = last_name
        self.payment_date = payment_date
        self.payment_amount = payment_amount
        self.paid = paid
```

Рисунок 6 – Класс Bill

Реализация класса ListPayer в свою очередь основывается на классе Bill, конструктор класса содержит список, в который должны попадать экземпляры класса Bill (рис.7).

```
def __init__(self, size):
    self.size = min(size, self.MAX_SIZE)
    self.count = 0
    self.payers = []
```

Рисунок 7 Конструктор ListPayer

Добавление, удаление и поиск записей отображён на рис. 8

```
def add_payer(self, payer):
    if self.count < self.size:
        self.payers.append(payer)
        self.count += 1
    else:
        print("Список полон")

def remove_payer(self, phone_number):
    for payer in self.payers:
        if payer.phone_number == phone_number:
            self.payers.remove(payer)
            self.count -= 1
            break

def find_by_phone(self, phone_number):
    for i, item in enumerate(self.payers):
        if item.phone_number == phone_number:
            self.show(i)

def find_by_last_name(self, last_name):
    for i, item in enumerate(self.payers):
        if item.last_name == last_name:
            self.show(i)

def total_payment_amount(self):
    return sum(payer.payment_amount for payer in self.payers)
```

Рисунок 8 – Методы

Пример использования отображен на рис. 8

```python
if __name__ == "__main__":
    # Пример использования:
    payer1 = Bill("1234567890", "Ivanov", datetime(2024, 10, 24), 100, True)
    payer2 = Bill("0987654321", "Petrov", datetime(2024, 10, 23), 150, False)

    list_payer = ListPayer(10)
    list_payer.add_payer(payer1)
    list_payer.add_payer(payer2)

    # Поиск по телефону
    list_payer.find_by_phone('0987654321')

    list_payer.find_by_last_name('Petrov')

    #Выводим общую сумму платежей
    print("Общаа сумма:", list_payer.total_payment_amount())

    payer21 = Bill("0987654321", "Ivanov", datetime(2024, 10, 24), 100, True)

    list_payer2 = ListPayer(10)
    list_payer2.add_payer(payer21)

    # Логические операции

    list_payer.union(list_payer2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
__конец вывода__

D:\Python-labs\4.2>python3 .\prog\idz2.py
__Начало вывода__
1 | 0987654321 Petrov 2024-10-23 150 False
__конец вывода__
__Начало вывода__
1 | 0987654321 Petrov 2024-10-23 150 False
__конец вывода__
Общаа сумма: 250
__Начало вывода__
0 | 0987654321 Petrov 2024-10-23 150 False
1 | 1234567890 Ivanov 2024-10-24 100 True
2 | 0987654321 Ivanov 2024-10-24 100 True
__конец вывода__

D:\Python-labs\4.2>
```

Рисунок 9 – Результат выполнения программы

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. 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?

Для перегрузки арифметических операций в Python используются следующие методы:

1) __add__(self, other) для сложения.
2) __sub__(self, other) для вычитания.
3) __mul__(self, other) для умножения.
4) __truediv__(self, other) для деления.

Для операций отношения используются методы:

1) __eq__(self, other) для проверки равенства.
2) __ne__(self, other) для проверки неравенства.
3) __lt__(self, other) для проверки "меньше чем".
4) __le__(self, other) для проверки "меньше или равно".
5) __gt__(self, other) для проверки "больше чем".
6) __ge__(self, other) для проверки "больше или равно".

3. В каких случаях будут вызваны следующие методы: add, iadd и radd? Приведите примеры.

Рассмотрим каждый по отдельности

__add__: Этот метод вызывается, когда используется оператор сложения. Например, a + b вызовет a.__add__(b).

iadd__: Этот метод вызывается для операции +=, которая является "in-place" сложением.

python

radd__: Этот метод вызывается, если первый операнд не поддерживает сложение, и Python пытается вызвать метод второго операнда. Например, b + a вызовет a.__radd__(b), если b не имеет метода __add__.

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 отвечает за выделение памяти под новый объект. __new__ возвращает новый экземпляр класса, тогда как __init__ инициализирует уже созданный экземпляр, устанавливая его начальное состояние.

5. Чем отличаются методы str и repr?

Метод __str__ предназначен для возвращения "читаемого" строкового представления объекта, которое будет использоваться, например, при вызове функции print(). Метод __repr__ возвращает более "официальное" строковое представление объекта, которое должно быть однозначным и, по возможности, позволять воссоздать объект при использовании функции eval().

Таким образом, __str__ используется для удобного отображения, а __repr__ — для отладки и разработки.

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