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 13**

Выполнил:
Мотовилов Вадим Борисович
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:
Воронкин Р.А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

**Тема:** Перегрузка операторов в языке Python

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

## Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее. Ссылка на гит: https://github.com/AkselSukub/OOP_2
2. Выполнил индивидуальное задание №1:

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

## Код индивидуального задания №1:

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

import math

class Pair:
    def __init__(self, first, second):
        if not isinstance(first, (int, float)) or not isinstance(second, (int, float)):
            raise ValueError("Katety должны быть числами.")
        if first <= 0 or second <= 0:
            raise ValueError("Katety должны быть положительными числами.")
        self.first = first
        self.second = second

    def hypotenuse(self):
        return math.sqrt(self.first ** 2 + self.second ** 2)

    def read(self):
        self.first = float(input("Введите длину первого катета (позитивное дробное число): "))
        self.second = float(input("Введите длину второго катета (позитивное дробное число): "))
        if self.first <= 0 or self.second <= 0:
            raise ValueError("Katety должны быть положительными числами.")

    def display(self):
        print(f"Katety: {self.first}, {self.second}. Гипотенуза: {self.hypotenuse()}")
```

```
def make_pair(first, second):
    try:
        return Pair(first, second)
    except ValueError as ve:
        print(ve)
        exit()


if __name__ == '__main__':
    try:
        katet1 = float(input("Введите длину первого катета: "))
        katet2 = float(input("Введите длину второго катета: "))
        pair = make_pair(katet1, katet2)
        pair.display()


        print("\nСейчас вы можете ввести новые значения катетов:")
        pair.read()
        pair.display()
    except ValueError as e:
        print(e)
```

4. Выполнил индивидуальное задание №2:

13. Информационная запись о книге в библиотеке содержит следующие поля: автор, название, год издания, издательство, цена. Для моделирования учетной карточки абонента реализовать класс Subscriber, содержащий фамилию абонента, его библиотечный номер и список взятых в библиотеке книг. Один элемент списка состоит из информационной записи о книге, даты выдачи, требуемой даты возврата и признака возврата. Реализовать методы добавления книг в список и удаления книг из него; метод поиска книг, подлежащих возврату; методы поиска по автору, издательству и году издания; метод вычисления стоимости всех подлежащих возврату книг. Реализовать операцию слияния двух учетных карточек, операцию пересечения и вычисления разности. Реализовать операцию генерации конкретного объекта Debt (долг), содержащего список книг, подлежащих возврату из объекта типа Subscriber.

## Код индивидуального задания №2:

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

from datetime import datetime, timedelta

class Book:
    def __init__(self, author, title, year, publisher, price):
        self.author = author
        self.title = title
```

```
        self.year = year
        self.publisher = publisher
        self.price = price

    def __str__(self):
        return f"{self.title} by {self.author} ({self.year}, {self.publisher}, {self.price} руб.)"

class Subscriber:
    MAX_BOOKS = 10  # Максимальное количество книг в списке

    def __init__(self, surname, library_number):
        self.surname = surname
        self.library_number = library_number
        self.books = []  # Список книг
        self.count = 0  # Текущий счетчик книг

    def size(self):
        """Возвращает максимальное количество книг."""
        return self.MAX_BOOKS

    def add_book(self, book, issue_date, return_date, returned=False):
        """Добавляет книгу в список, если не превышен лимит."""
        if self.count >= self.MAX_BOOKS:
            print("Достигнуто максимальное количество книг.")
            return
        self.books.append({
            'book': book,
            'issue_date': issue_date,
            'return_date': return_date,
            'returned': returned
        })
        self.count += 1

    def remove_book(self, index):
        """Удаляет книгу по индексу."""
        if 0 <= index < self.count:
            del self.books[index]
            self.count -= 1
        else:
            print("Некорректный индекс.")

    def find_due_books(self):
        """Находит книги, которые подлежат возврату."""
        due_books = []
        today = datetime.now().date()
        for entry in self.books:
            if not entry['returned'] and entry['return_date'] < today:
                due_books.append(entry)
        return due_books

    def find_by_author(self, author):
        """Находит книги по автору."""
        return [entry for entry in self.books if entry['book'].author == author]

    def find_by_publisher(self, publisher):
        """Находит книги по издательству."""
        return [entry for entry in self.books if entry['book'].publisher == publisher]

    def find_by_year(self, year):
        """Находит книги по году издания."""
        return [entry for entry in self.books if entry['book'].year == year]

    def total_due_cost(self):
```

```
        """Вычисляет стоимость всех подлежащих возврату книг."""
        total_cost = sum(entry['book'].price for entry in self.find_due_books())
        return total_cost

    def merge(self, other):
        """Сливает две учетные карточки."""
        for entry in other.books:
            if self.count < self.MAX_BOOKS:
                self.books.append(entry)
                self.count += 1

    def intersection(self, other):
        """Возвращает пересечение двух учетных карточек."""
        common_books = []
        for entry in self.books:
            if entry in other.books:
                common_books.append(entry)
        return common_books

    def difference(self, other):
        """Возвращает разность двух учетных карточек."""
        unique_books = [entry for entry in self.books if entry not in other.books]
        return unique_books

    def __getitem__(self, index):
        """Перегрузка индексирования для доступа к книгам."""
        if 0 <= index < self.count:
            return self.books[index]
        raise IndexError("Индекс вне диапазона.")

    def __str__(self):
        """Возвращает строковое представление объекта."""
        return f"Абонент: {self.surname}, Номер: {self.library_number}, Книги: {self.count}"

class Debt:
    def __init__(self, subscriber):
        self.subscriber = subscriber
        self.due_books = subscriber.find_due_books()

    def __str__(self):
        return f"Долг абонента {self.subscriber.surname}: {len(self.due_books)} книг."

def input_book():
    """Функция для ввода данных о книге."""
    author = input("Введите автора книги: ")
    title = input("Введите название книги: ")
    year = int(input("Введите год издания: "))
    publisher = input("Введите издательство: ")
    price = float(input("Введите цену книги: "))
    return Book(author, title, year, publisher, price)

def main():
    surname = input("Введите фамилию абонента: ")
    library_number = input("Введите библиотечный номер абонента: ")
    subscriber = Subscriber(surname, library_number)

    while True:
        print("\nВыберите действие:")
        print("1. Добавить книгу")
        print("2. Удалить книгу")
        print("3. Найти книги по автору")
        print("4. Найти книги по издательству")
        print("5. Найти книги по году издания")
```

```
        print("6. Показать книги, подлежащие возврату")
        print("7. Показать общую стоимость долгов")
        print("8. Выход")

        choice = input("Ваш выбор: ")

        if choice == '1':
            book = input_book()
            issue_date = datetime.now().date()
            return_date = issue_date + timedelta(days=30)  # Предположим, что срок займа 30 дней
            subscriber.add_book(book, issue_date, return_date)
            print("Книга добавлена.")

        elif choice == '2':
            index = int(input("Введите индекс книги для удаления (0 - {}): ".format(subscriber.count - 1)))
            subscriber.remove_book(index)
            print("Книга удалена.")

        elif choice == '3':
            author = input("Введите автора для поиска: ")
            found_books = subscriber.find_by_author(author)
            for entry in found_books:
                print(entry['book'])

        elif choice == '4':
            publisher = input("Введите издательство для поиска: ")
            found_books = subscriber.find_by_publisher(publisher)
            for entry in found_books:
                print(entry['book'])

        elif choice == '5':
            year = int(input("Введите год издания для поиска: "))
            found_books = subscriber.find_by_year(year)
            for entry in found_books:
                print(entry['book'])

        elif choice == '6':
            due_books = subscriber.find_due_books()
            if due_books:
                for entry in due_books:
                    print(f"Книга подлежит возврату: {entry['book']}")
            else:
                print("Нет книг, подлежащих возврату.")

        elif choice == '7':
            total_cost = subscriber.total_due_cost()
            print(f"Общая стоимость долгов: {total_cost} руб.")

        elif choice == '8':
            print("Выход из программы.")
            break

        else:
            print("Некорректный выбор, попробуйте снова.")

if __name__ == '__main__':
    main()
```

**Ответы на контрольные вопросы:**

**1. Какие средства существуют в Python для перегрузки операций?**

В Python перегрузка операций осуществляется с помощью магических методов, которые определяются внутри класса. Они позволяют переопределить поведение стандартных операторов для работы с объектами этого класса. Перегрузка операторов − один из способов реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе.

**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 “inplace” сложением.

__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.