Министерство науки и высшего образования

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

«Уфимский университет науки и технологий»

Кафедра АСУ

Отчет по лабораторным работам

Дисциплина: «Объектно-ориентированное моделирование и программирование»

Выполнил: ст. гр. ПИ-215Бз

Швецов А.Е.

Проверил: канд. техн. наук, доцент

Кромина Л.А.

Уфа – 2023

# **Лабораторная работа №1**

**Конструктор и деструктор**

**Цель:** освоить применение конструктора и деструктора в языке программирования «Python»

**Задание**

Напишите программу по следующему описанию:

1. Создайте класс «Сотрудник», конструктор \_\_init\_\_() которого принимает три параметра (не учитывая self) – имя (name), фамилию (lastname) и разряд сотрудника (discharge). Разряд имеет значение, заданное по умолчанию, равное 6.

2. У класса «Сотрудник» создайте метод conclusion, позволяющий выводить строку, содержащую всю информацию о сотруднике.

3. У класса «Сотрудник» создайте деструктор \_\_del\_\_(), который выводит на экран фразу «Всего доброго, …» (вместо многоточия должны выводиться имя и фамилия сотрудника).

4. Создайте три объекта класса «Сотрудник»: Иван Иванов, имеющий 3 разряд; Петр Петров, имеющий 5 разряд; Петр Сидоров, имеющий 2 разряд.

5. Применяя деструктор, а также условные операторы, попрощайтесь с сотрудником, имеющим минимальное значение разряда.

**Программный код**

*# Инициализация класса Работник*

class Employee:

    def \_\_init\_\_(*self*, *name*, *surename*, *discharge*=6):

*self*.name = *name*

*self*.surename = *surename*

*self*.discharge = *discharge*

*# Создание метода для вывода всей информации о сотруднике*

    def conclusion(*self*):

        print(

            "\n",

            "Имя =",

*self*.name,

            "\n",

            "Фамилия =",

*self*.surename,

            "\n",

            "Разряд =",

*self*.discharge,

        )

*# Создание деструктора*

    def \_\_del\_\_(*self*):

        print("\n", "Всего доброго,", *self*.name, *self*.surename)

*# Информация о сотрудниках*

worker1 = Employee("Иван", "Иванов", 3)

worker2 = Employee("Петр", "Петров", 5)

worker3 = Employee("Петр", "Сидоров", 2)

*# Вывод всей информации о всех сотрудниках*

worker1.conclusion()

worker2.conclusion()

worker3.conclusion()

*# Прощаемся с сотрудником, имеющим минимальное значение разряда*

*if* worker1.discharge < worker2.discharge *and* worker1.discharge < worker3.discharge:

*del* worker1

*elif* worker2.discharge < worker1.discharge *and* worker2.discharge < worker3.discharge:

*del* worker2

*elif* worker3.discharge < worker1.discharge *and* worker3.discharge < worker2.discharge:

*del* worker3

input()

**Результат работы программы**

lexot@DESKTOP-EOIUDK6 MINGW64 ~

$ C:/Users/lexot/AppData/Local/Programs/Python/Python310/python.exe

d:/Github/OOMiP-labs/laba.py

 Имя = Иван

 Фамилия = Иванов

 Разряд = 3

 Имя = Петр

 Фамилия = Петров

 Разряд = 5

 Имя = Петр

 Фамилия = Сидоров

 Разряд = 2

 Всего доброго, Петр Сидоров

**Вывод о проделанной лабораторной работе:** я на практике освоил применение конструктора и деструктора в языке программирования «Python». Данная работа помогла мне углубить знания по основам объектно-ориентированного программирования.

# **Лабораторная работа №2**

**Перегрузка операторов**

**Цель:** освоить применение перегрузки операторов в языке программирования «Python»

**Задание**

1. Создайте класс «snowflakes», в конструкторе класса инициируйте поле, содержащее количество снежинок «number\_of\_snowflakes».

2. В классе пропишите следующие методы перегрузки арифметических операторов:

1) \_\_add\_\_() – сложение;

2) \_\_ sub \_\_() – вычитание;

3) \_\_ mul \_\_() – умножение;

4) \_\_ truediv\_\_() – деление (деление «/», при этом необходимо организовать округление значения до целого числа);

5) \_\_ floordiv\_\_() – деление (целочисленное деление «//»);

6) \_\_ mod \_\_ () – остаток от деления (%).

3. Создайте объект класса «snowflakes» – A;

4. Применяя функцию «print», проверьте работоспособность перегрузки операторов.

**Программный код**

*# Инициализация класса Снежинки*

class Snowflakes:

    def \_\_init\_\_(*self*, *number\_of\_snowflakes*):

*self*.number\_of\_snowflakes = *number\_of\_snowflakes*

*# Создание перегрузки оператора Сложения*

    def \_\_add\_\_(*self*, *other*):

*return* *self*.number\_of\_snowflakes + *other*.number\_of\_snowflakes

*# Создание перегрузки оператора Вычитания*

    def \_\_sub\_\_(*self*, *other*):

*return* *self*.number\_of\_snowflakes - *other*.number\_of\_snowflakes

*# Создание перегрузки оператора Умножения*

    def \_\_mul\_\_(*self*, *other*):

*return* *self*.number\_of\_snowflakes \* *other*.number\_of\_snowflakes

*# Создание перегрузки оператора Деления*

    def \_\_truediv\_\_(*self*, *other*):

*return* round(*self*.number\_of\_snowflakes / *other*.number\_of\_snowflakes)

*# Создание перегрузки оператора Целочисленного деления*

    def \_\_floordiv\_\_(*self*, *other*):

*return* *self*.number\_of\_snowflakes // *other*.number\_of\_snowflakes

*# Создание перегрузки оператора Взятия остатка*

    def \_\_mod\_\_(*self*, *other*):

*return* *self*.number\_of\_snowflakes % *other*.number\_of\_snowflakes

*# Создание объекта А*

A = Snowflakes(10)

*# Тестирование перегрузки операторов*

print("Сложение -", A + Snowflakes(5))

print("Вычитание -", A - Snowflakes(3))

print("Умножение -", A \* Snowflakes(2))

print("Деление -", A / Snowflakes(5))

print("Целочисленное деление -", A // Snowflakes(3))

print("Взятие остатка -", A % Snowflakes(3))

**Результат работы программы**

lexot@DESKTOP-EOIUDK6 MINGW64 ~

$ C:/Users/lexot/AppData/Local/Programs/Python/Python310/python.exe

d:/Github/OOMiP-labs/laba2.py

Сложение - 15

Вычитание - 7

Умножение - 20

Деление - 2

Целочисленное деление - 3

Взятие остатка - 1

**Вывод о проделанной лабораторной работе:** Я на практике освоил применение перегрузки операторов в языке программирования «Python». Данная работа помогла мне углубить знания по основам объектно-ориентированного программирования.