

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 №4.2
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Бересланов Рамазан
3 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., доцент департамента
цифровых, робототехнических систем и
электроники

(подпись)

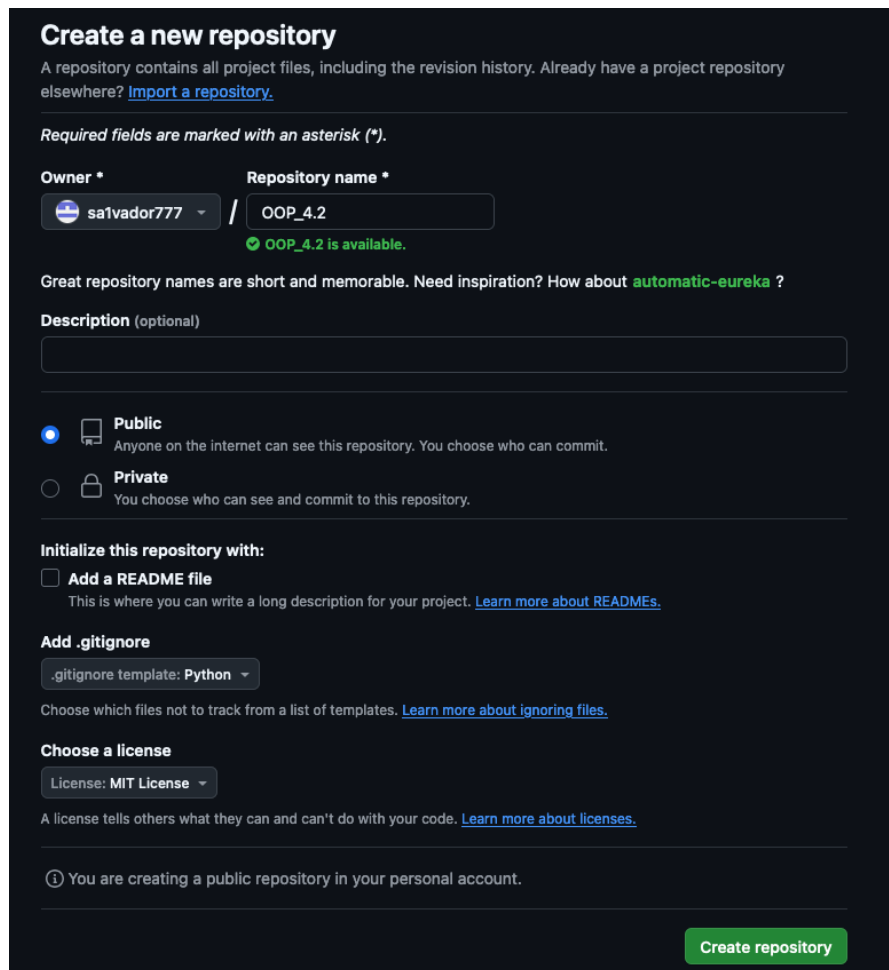
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

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

Ход работы:

1. Изучил теоретический материал.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner * salvador777 / Repository name * OOP_4.2
✔ OOP_4.2 is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-eureka](#) ?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

[Create repository](#)

Рисунок 1 – Создание репозитория

3. Склонировал репозиторий на ПК.

```
> git clone https://github.com/salvador777/OOP_4.2.git
Cloning into 'OOP_4.2'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.
```

Рисунок 2 – Клонирование репозитория

4. Инициализировал модель ветвления git flow.

```
> git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [/Users/blessedaxe/programming/ncfu/ooP_4.2/.git/hooks]
```

Рисунок 3 – Инициализация git-flow

5. Создал виртуальное окружение poetry

```
> poetry init

This command will guide you through creating your pyproject.toml config.

Package name [oop_4.2]:
Version [0.1.0]:
Description []:
Author [salvador77 <uncrowned.blessed@gmail.com>, n to skip]:
License []:
Compatible Python versions [^3.12]:
```

Рисунок 4 – Создание окружения poetry

```
> poetry shell
Creating virtualenv oop-4-2-BNhDMHF8-py3.12 in /Users/blessedaxe/Library/Caches/pypoetry/virtualenvs
Spawning shell within /Users/blessedaxe/Library/Caches/pypoetry/virtualenvs/ooP-4-2-BNhDMHF8-py3.12
> emulate bash -c './Users/blessedaxe/Library/Caches/pypoetry/virtualenvs/ooP-4-2-BNhDMHF8-py3.12/bin/activate'
```

Рисунок 5 – Активация окружения

1) Задание 1.

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

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 class Pair:
5     def __init__(self, first, second):
6         if not isinstance(first, float):
7             raise ValueError("Поле first должно быть дробным числом.")
8         if not isinstance(second, int):
9             raise ValueError("Поле second должно быть целым числом.")
10        self.first = first
11        self.second = second
12
13    def read(self):
14        try:
15            self.first = float(input("Введите значение first (дробное число): "))
16            self.second = int(input("Введите значение second (целое число): "))
17        except ValueError as e:
18            print(f"Ошибка ввода: {e}")
19            return
20
21    def display(self):
22        print(f"Первое число: {self.first}, Второе число: {self.second}")
23
24    def power(self):
25        return self.first ** self.second
26
27 def make_pair(first, second):
28     try:
29         return Pair(first, second)
30     except ValueError as e:
31         print(f"Ошибка создания пары: {e}")
32         return None

```

Рисунок 6 – Основной код индивидуального задания №1 лабораторной работы 4.1

```

27 # Перегрузка оператора сложения
28 def __add__(self, other):
29     if isinstance(other, Pair):
30         return Pair(self.first + other.first, self.second + other.second)
31     raise ValueError("Операнд должен быть объектом Pair.")
32
33 # Перегрузка оператора вычитания
34 def __sub__(self, other):
35     if isinstance(other, Pair):
36         return Pair(self.first - other.first, self.second - other.second)
37     raise ValueError("Операнд должен быть объектом Pair.")
38
39 # Перегрузка оператора сравнения (по полю first)
40 def __eq__(self, other):
41     if isinstance(other, Pair):
42         return self.first == other.first and self.second == other.second
43     return False
44
45 def __lt__(self, other):
46     if isinstance(other, Pair):
47         return (self.first, self.second) < (other.first, other.second)
48     return False
49
50 # Приведение к строке для удобного вывода
51 def __str__(self):
52     return f"({self.first}, {self.second})"

```

Рисунок 7 – Добавленные перегрузки операторов

2) Задание 2. Вариант - 1

Создать класс `Decimal` для работы с беззнаковыми целыми десятичными числами, используя для представления числа список из 100 элементов типа `int`, каждый из которых является десятичной цифрой. Младшая цифра имеет меньший индекс (единицы — в нулевом элементе списка). Реальный размер списка задается как аргумент конструктора инициализации. Реализовать арифметические операции, аналогичные встроенным для целых и операции сравнения.

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. В тех задачах, где возможно, реализовать конструктор инициализации строкой.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 class Decimal:
6     MAX_SIZE = 100
7
8     def __init__(self, size):
9         """Инициализация списка с цифрами числа. Размер не может превышать MAX_SIZE."""
10        if size > self.MAX_SIZE or size <= 0:
11            raise ValueError(f"Размер числа должен быть в пределах от 1 до {self.MAX_SIZE}.")
12        self.digits = [0] * size
13        self.size = size
14        self.count = 0
15
16    def __repr__(self):
17        """Строковое представление числа (старшие разряды идут первыми)."""
18        return ''.join(map(str, reversed(self.digits)))
19
20    def set_number(self, number):
21        """Устанавливает число в виде списка цифр."""
22        number_str = str(number)[::-1]
23        if len(number_str) > self.size:
24            raise ValueError(f"Число слишком большое для заданного размера: {self.size}")
25        self.digits = [int(digit) for digit in number_str] + [0] * (self.size - len(number_str))
26        self.count = len(number_str)
27
28    def get_number(self):
29        """Возвращает число как строку (старшие разряды идут первыми)."""
30        return ''.join(map(str, reversed(self.digits))).rstrip('0') or '0'
```

Рисунок 8 – Основной код второго индивидуального задания

```

# Перегрузка оператора сложения
def __add__(self, other):
    if isinstance(other, Decimal):
        result = Decimal(self.size)
        carry = 0
        for i in range(self.size):
            sum_digits = self.digits[i] + other.digits[i] + carry
            result.digits[i] = sum_digits % 10
            carry = sum_digits // 10
        result.count = max(self.count, other.count)
        return result
    raise ValueError("Числа должны быть одинакового размера и типа Decimal.")

# Перегрузка оператора вычитания
def __sub__(self, other):
    if isinstance(other, Decimal):
        result = Decimal(self.size)
        borrow = 0
        for i in range(self.size):
            diff_digits = self.digits[i] - other.digits[i] - borrow
            if diff_digits < 0:
                diff_digits += 10
                borrow = 1
            else:
                borrow = 0
            result.digits[i] = diff_digits
        result.count = max(self.count, other.count)
        return result
    raise ValueError("Числа должны быть одинакового размера и типа Decimal.")

# Перегрузка оператора умножения
def __mul__(self, other):
    if isinstance(other, Decimal):
        result = Decimal(self.size)
        temp_result = [0] * (2 * self.size)
        for i in range(self.size):
            carry = 0
            for j in range(self.size):
                mul_digits = self.digits[i] * other.digits[j] + temp_result[i + j] + carry
                temp_result[i + j] = mul_digits % 10
                carry = mul_digits // 10
            # Обрезаем лишние разряды
            result.digits = temp_result[:self.size]
            result.count = self.count + other.count
        return result
    raise ValueError("Числа должны быть одинакового размера и типа Decimal.")

```

Рисунок 9 – Перегрузка основных операторов

```

# Операция сравнения: ==
def __eq__(self, other):
    if isinstance(other, Decimal):
        return self.digits == other.digits
    return False

# Операция сравнения: <
def __lt__(self, other):
    if isinstance(other, Decimal):
        return self.get_number() < other.get_number()
    return False

# Операция сравнения: >
def __gt__(self, other):
    if isinstance(other, Decimal):
        return self.get_number() > other.get_number()
    return False

# Приведение к строке для удобного отображения
def __str__(self):
    return self.get_number()

```

Рисунок 10 – Перегрузка операторов сравнения и приведение к строке

```

if __name__ == "__main__":
    # Демонстрация работы класса Decimal

    # Создаем два числа
    dec1 = Decimal(10)
    dec2 = Decimal(10)

    # Устанавливаем значения
    dec1.set_number(12345)
    dec2.set_number(6789)

    # Вывод чисел
    print("dec1:", dec1)
    print("dec2:", dec2)

    # Демонстрация операций
    print("\nРезультат сложения:", dec1 + dec2)
    print("Результат вычитания:", dec1 - dec2)
    print("Результат умножения:", dec1 * dec2)

    # Сравнение
    print("\ndec1 == dec2:", dec1 == dec2)
    print("dec1 > dec2:", dec1 > dec2)
    print("dec1 < dec2:", dec1 < dec2)

    # Проверка значений size и count
    print(f"\nРазмер dec1 (size): {dec1.size}")
    print(f"Количество активных цифр в dec1 (count): {dec1.count}")

```

```
> python src/task2.py
dec1: 12345
dec2: 6789

Результат сложения: 19134
Результат вычитания: 5556
Результат умножения: 83810205

dec1 == dec2: False
dec1 > dec2: False
dec1 < dec2: True

Размер dec1 (size): 10
Количество активных цифр в dec1 (count): 5
```

Рисунки 11, 12 – Демонстрация работы программы

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

В Python перегрузка операций осуществляется с помощью **магических методов** (или методов "dunder" — double underscore). Эти методы имеют специальный синтаксис и позволяют изменять поведение стандартных операторов для пользовательских классов. Например:

- `__add__(self, other)` — перегрузка оператора сложения +
- `__sub__(self, other)` — перегрузка оператора вычитания -
- `__mul__(self, other)` — перегрузка оператора умножения *

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

Методы для перегрузки арифметических операций:

- `__add__(self, other)` — сложение

- `__sub__(self, other)` — вычитание
- `__mul__(self, other)` — умножение
- `__truediv__(self, other)` — деление
- `__floordiv__(self, other)` — целочисленное деление
- `__mod__(self, other)` — остаток от деления
- `__pow__(self, other)` — возведение в степень

Методы для перегрузки операций отношения:

- `__eq__(self, other)` — равно `==`
- `__ne__(self, other)` — не равно `!=`
- `__gt__(self, other)` — больше `>`
- `__lt__(self, other)` — меньше `<`
- `__ge__(self, other)` — больше или равно `>=`
- `__le__(self, other)` — меньше или равно `<=`

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

- `__add__(self, other)`: вызывается при использовании оператора `+` между двумя объектами. Например: **`result = obj1 + obj2`**
- `__iadd__(self, other)`: вызывается при использовании оператора `+=`. Например: **`obj1 += obj2`**
- `__radd__(self, other)`: вызывается, когда оператор `+` применяется с объектом, у которого нет метода `__add__`. Например, если `other` (правый операнд) имеет метод `__radd__`, он будет вызван, если `obj1` (левый операнд) не имеет метода `__add__`.

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

- Метод `__new__` предназначен для создания нового экземпляра класса. Он вызывается перед `__init__` и возвращает новый объект. Используется, когда нужно управлять созданием объекта, например, в случае с метаклассами или синглтонами.
- Метод `__init__` — это конструктор, который инициализирует уже созданный объект. В `__init__` задаются атрибуты экземпляра.

В общем, `__new__` отвечает за создание, а `__init__` — за инициализацию.

5. Чем отличаются методы `__str__` и `__repr__`?

- Метод `__str__` предназначен для создания строкового представления объекта, которое будет читабельно для пользователя. Используется, когда вызывается `print()` или `str()`.
- Метод `__repr__` предназначен для создания представления объекта, которое может быть использовано для восстановления объекта. Обычно это более детальное и формальное представление, подходящее для разработчиков. Используется при вызове `repr()` и в интерактивной среде Python.