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»

Выполнил:
Кожуховский Виктор Андреевич
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 Роман Александрович

_____
(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. Изучил теоретический материал работы.

2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.

3. Выполнил клонирование созданного репозитория.

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE.

5. Организовал свой репозиторий в соответствие с моделью ветвления git-flow.

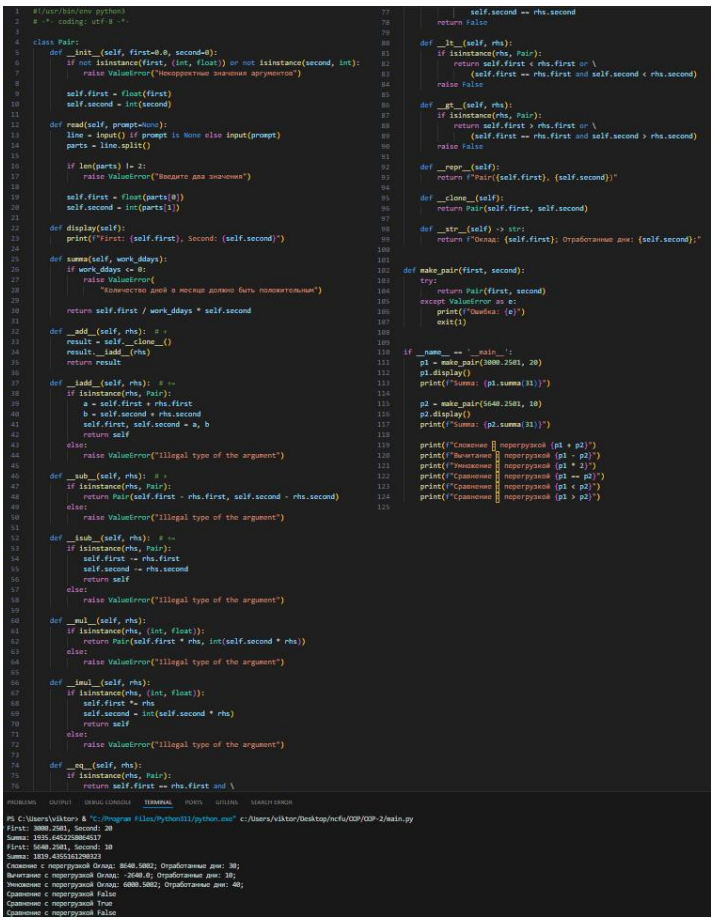
6. Создал проект в папке репозитория.

7. Проработал примеры лабораторной работы.

8. Выполнил индивидуальное задание для варианта 14.

Задание 1

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



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

class Pair:
    def __init__(self, first=0.0, second=0):
        if not isinstance(first, (int, float)) or not isinstance(second, int):
            raise ValueError("Некорректные значения аргументов")

        self.first = float(first)
        self.second = int(second)

    def read(self, prompt=None):
        line = input() if prompt is None else input(prompt)
        parts = line.split()

        if len(parts) != 2:
            raise ValueError("Введите два значения")

        self.first = float(parts[0])
        self.second = int(parts[1])

    def display(self):
        print(f"First: {self.first}, Second: {self.second}")

    def summa(self, work_ddays):
        if work_ddays <= 0:
            raise ValueError(
                "Количество дней в месяце должно быть положительным")

        return self.first / work_ddays * self.second

    def __add__(self, rhs):  # +
        result = self.__clone__()
        result.__iadd__(rhs)
        return result

    def __iadd__(self, rhs):  # +=
        if isinstance(rhs, Pair):
            a = self.first + rhs.first
            b = self.second + rhs.second
            self.first, self.second = a, b
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __sub__(self, rhs):  # +
        if isinstance(rhs, Pair):
            return Pair(self.first - rhs.first, self.second - rhs.second)
        else:
            raise ValueError("Illegal type of the argument")

    def __isub__(self, rhs):  # +=
        if isinstance(rhs, Pair):
            self.first -= rhs.first
            self.second -= rhs.second
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __mul__(self, rhs):
        if isinstance(rhs, (int, float)):
            return Pair(self.first * rhs, int(self.second * rhs))
        else:
            raise ValueError("Illegal type of the argument")

    def __imul__(self, rhs):
        if isinstance(rhs, (int, float)):
            self.first *= rhs
            self.second = int(self.second * rhs)
            return self
        else:
            raise ValueError("Illegal type of the argument")

    def __eq__(self, rhs):
        if isinstance(rhs, Pair):
            return self.first == rhs.first and \
                   self.second == rhs.second
        return False

    def __lt__(self, rhs):
        if isinstance(rhs, Pair):
            return self.first < rhs.first or \
                   (self.first == rhs.first and self.second < rhs.second)
        raise False

    def __gt__(self, rhs):
        if isinstance(rhs, Pair):
            return self.first > rhs.first or \
                   (self.first == rhs.first and self.second > rhs.second)
        raise False

    def __repr__(self):
        return f"Pair({self.first}, {self.second})"

    def __clone__(self):
        return Pair(self.first, self.second)

    def __str__(self) -> str:
        return f"Оклад: {self.first}; Отработанные дни: {self.second};"


def make_pair(first, second):
    try:
        return Pair(first, second)
    except ValueError as e:
        print(f"Ошибка: {e}")
        exit(1)


if __name__ == '__main__':
    p1 = make_pair(3000.2501, 20)
    p1.display()
    print(f"Summa: {p1.summa(31)}")

    p2 = make_pair(5640.2501, 10)
    p2.display()
    print(f"Summa: {p2.summa(31)}")

    print(f"Сложение с перегрузкой {p1 + p2}")
    print(f"Вычитание с перегрузкой {p1 - p2}")
    print(f"Умножение с перегрузкой {p1 * 2}")
    print(f"Сравнение с перегрузкой {p1 == p2}")
    print(f"Сравнение с перегрузкой {p1 < p2}")
    print(f"Сравнение с перегрузкой {p1 > p2}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SEARCH ERROR

PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" c:/Users/viktor/Desktop/ncfu/ООП/ООП-2/main.py
First: 3000.2501, Second: 20
Summa: 1935.6452258064517
First: 5640.2501, Second: 10
Summa: 1819.4355161290323
Сложение с перегрузкой Оклад: 8640.5002; Отработанные дни: 30;
Вычитание с перегрузкой Оклад: -2640.0; Отработанные дни: 10;
Умножение с перегрузкой Оклад: 6000.5002; Отработанные дни: 40;
Сравнение с перегрузкой False
Сравнение с перегрузкой True
Сравнение с перегрузкой False
```

Рисунок 1. Код решения индивидуального задания 1 и его выполнение

Задание 2

Информационная запись о файле в каталоге содержит поля: имя файла, расширение, дата и время создания, атрибуты «только чтение», «скрытый», «системный», размер файла на диске. Для моделирования каталога реализовать класс Directory, содержащий название родительского каталога, количество файлов в каталоге, список файлов в каталоге. Один элемент списка включает в себя информационную запись о файле, дату последнего изменения, признак выделения и признак удаления. Реализовать методы добавления файлов в каталог и удаления файлов из него; метод поиска файла по имени, по расширению, по дате создания; метод вычисления полного объема каталога. Реализовать операцию объединения и операцию пересечения каталогов. Реализовать операцию генерации конкретного объекта Group (группа), содержащего список файлов, из объекта типа Directory. Должна быть возможность выбирать группу файлов по признаку удаления, по атрибутам, по дате создания (до или после), по объему (меньше или больше).

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.

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

from datetime import datetime


class FileInfo:
    def __init__(self, name, extension, creation_date,
                 read_only, hidden, system, size):
        self.name = name
        self.extension = extension
        self.creation_date = creation_date
        self.read_only = read_only
        self.hidden = hidden
        self.system = system
        self.size = size
        self.last_modified = datetime.now()
        self.selected = False
        self.deleted = False


class Directory:
    MAX_FILES = 1000  # Максимально возможный размер списка задан константой.

    def __init__(self, parent_dir, size=MAX_FILES):
        self.parent_dir = parent_dir
        self.size = size
        self.count = 0
        self.files = []

    def add_file(self, file_info):  # Добавление файлов в каталог
        if self.count < self.size:
            self.files.append(file_info)
            self.count += 1
        else:
            print("Directory is full")

    def remove_file(self, file_name):  # Удаления файлов из каталога
        self.files = [f for f in self.files if f.name != file_name]
        self.count = len(self.files)

    def find_file(self, **kwargs):  # Поиск файла
        return [f for f in self.files if all(
            getattr(f, k) == v for k, v in kwargs.items())]

    def total_size(self):  # метод вычисления полного объема каталога
        return sum(f.size for f in self.files)

    def __add__(self, other):  # операция объединения каталогов
        new_dir = Directory(self.parent_dir, self.size + other.size)
        new_dir.files = self.files + other.files
        new_dir.count = len(new_dir.files)
        return new_dir

    def __and__(self, other):
        new_dir = Directory(self.parent_dir)
        new_dir.files = [f for f in self.files if any(
            f.name == of.name and
            f.extension == of.extension for of in other.files)]
        new_dir.count = len(new_dir.files)
        return new_dir

    def generate_group(self, **kwargs):
        return Group([f for f in self.files if all(
            (callable(v) and v(getattr(f, k))) or (getattr(f, k) == v)
            for k, v in kwargs.items())])

    def __getitem__(self, index):
        return self.files[index]

    def __len__(self):
        return self.count

    def size(self):
        return self.size


class Group:
    def __init__(self, files):
        self.files = files


if __name__ == "__main__":
    dir1 = Directory("C:/Users/viktor/Desktop/ncfu/OOP/trash")
    dir1.add_file(FileInfo("file1", "txt", datetime.now(), False,
                  False, False, 100))  # read_only, hidden, system, size
    dir1.add_file(FileInfo("file2", "doc", datetime.now(),
                  True, False, False, 200))

    found_files = dir1.find_file(name="file1")
    print(f"Found files: {[f.name for f in found_files]}")

    total_size = dir1.total_size()
    print(f"Total size: {total_size}")

    group = dir1.generate_group(extension="txt")
    print(f"Group files: {[f.name for f in group.files]}")

    first_file = dir1[0]
    print(f"First file: {first_file.name}")

    last_file = dir1[-1]
    print(f"Last file: {last_file.name}")

    print(f"Directory size: {dir1.size}")
    print(f"Number of files: {len(dir1)}")

    # Объединение каталогов
    dir2 = Directory("C:/Users/viktor/Desktop/ncfu/OOP/trash2")
    dir2.add_file(FileInfo("file3", "pdf", datetime.now(),
                  False, False, False, 300))
    combined_dir = dir1 + dir2
    print(f"Combined directory files: {[f.name for f in combined_dir.files]}")

    # Пересечение каталогов
    dir3 = Directory("C:/Users/viktor/Desktop/ncfu/OOP/trash3")
    dir3.add_file(FileInfo("file1", "txt", datetime.now(),
                  False, False, False, 100))
    intersected_dir = dir1 & dir3
    print(
        "Intersected directory files:"
        f"{[f.name for f in intersected_dir.files]}")

    # Генерация группы по различным критериям
    group_by_extension = dir1.generate_group(extension="txt")
    print(f"Group by extension: {[f.name for f in group_by_extension.files]}")

    group_by_size = dir1.generate_group(size=lambda x: x > 150)
    print(f"Group by size > 150: {[f.name for f in group_by_size.files]}")

    group_by_date = dir1.generate_group(
        creation_date=lambda x: x >= datetime(2024, 10, 10))
    print("Group by date >= 2024.10.10:"
          f"{[f.name for f in group_by_date.files]}")

    group_by_attribute = dir1.generate_group(read_only=True)
    print(
        "Group by read_only attribute:"
        f"{[f.name for f in group_by_attribute.files]}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SEARCH ERROR

Found files: ['file1']
Total size: 300
Group files: ['file1']
First file: file1
Last file: file2
Directory size: 1000
Number of files: 2
Combined directory files: ['file1', 'file2', 'file3']
Intersected directory files:['file1']
Group by extension: ['file1']
Group by size > 150: ['file2']
Group by date >= 2024.10.10:['file1', 'file2']
Group by read_only attribute:['file2']
```

Рисунок 2. Код решения индивидуального задания 2 и его выполнение

9. Зафиксировал сделанные изменения в репозитории.

10. Выполнил слияние ветки для разработки с веткой master/main.

11. Отправил сделанные изменения на сервер GitHub.

Ссылка: https://github.com/Viktorkozh/OOP-2

Контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

__new__(cls[, ...]) — управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу __init__ .

__init__(self[, ...]) - как уже было сказано выше, конструктор.

__del__(self) - вызывается при удалении объекта сборщиком мусора.

__repr__(self) - вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.

__str__(self) - вызывается функциями str, print и format. Возвращает строковое представление объекта.

__bytes__(self) - вызывается функцией bytes при преобразовании к байтам.

__format__(self, format_spec) - используется функцией format (а также методом format у строк).

__lt__(self, other) - x < y вызывает x.__lt__(y) .

__le__(self, other) - x ≤ y вызывает x.__le__(y) .

__eq__(self, other) - x == y вызывает x.__eq__(y) .

__ne__(self, other) - x != y вызывает x.__ne__(y) .

__gt__(self, other) - x > y вызывает x.__gt__(y) .

__ge__(self, other) - x ≥ y вызывает x.__ge__(y) .

__hash__(self) - получение хэш-суммы объекта, например, для добавления в словарь.

__bool__(self) - вызывается при проверке истинности. Если этот метод не определён, вызывается метод __len__ (объекты, имеющие ненулевую длину, считаются истинными).

__getattr__(self, name) - вызывается, когда атрибут экземпляра класса не найден в обычных местах (например, у экземпляра нет метода с таким названием).

__setattr__(self, name, value) - назначение атрибута.

__delattr__(self, name) - удаление атрибута (del obj.name).

__call__(self[, args...]) - вызов экземпляра класса как функции.

__len__(self) - длина объекта.

__getitem__(self, key) - доступ по индексу (или ключу).

__setitem__(self, key, value) - назначение элемента по индексу.

__delitem__(self, key) - удаление элемента по индексу.

__iter__(self) - возвращает итератор для контейнера.

__reversed__(self) - итератор из элементов, следующих в обратном порядке.

__contains__(self, item) - проверка на принадлежность элемента контейнеру (item in self).

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 ^ y).

__or__(self, other) - битовое ИЛИ (x | y).

__radd__(self, other) ,

__rsub__(self, other) ,

__rmul__(self, other) ,

__rtruediv__(self, other) ,

__rfloordiv__(self, other) ,

__rmod__(self, other) ,

__rdivmod__(self, other) ,

__rpow__(self, other) ,

__rlshift__(self, other) ,

__rrshift__(self, other) ,

__rand__(self, other) ,

__rxor__(self, other) ,

__ror__(self, other) - делают то же самое, что и арифметические операторы, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

__iadd__(self, other) - += .

__isub__(self, other) - -= .

__imul__(self, other) - *= .

__itruediv__(self, other) - /= .

__ifloordiv__(self, other) - //= .

__imod__(self, other) - %= .

__ipow__(self, other[, modulo]) - **= .

__ilshift__(self, other) - <<= .

__irshift__(self, other) - >>= .

__iand__(self, other) - &= .

__ixor__(self, other) - ^= .

__ior__(self, other) - |= .

__neg__(self) - унарный -.

__pos__(self) - унарный +.

__abs__(self) - модуль (abs()).

__invert__(self) - инверсия (~).

__complex__(self) - приведение к complex.

__int__(self) - приведение к int.

__float__(self) - приведение к float.

__round__(self[, n]) - округление.

__enter__(self) , __exit__(self, exc_type, exc_value, traceback) – реализация менеджеров контекста, используемый оператором with.

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

Приведите примеры.

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) .

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

__new__(cls[, ...]) — управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу __init__ .

5. Чем отличаются методы __str__ и __repr__ ?

__str__(self) - вызывается функциями str, print и format. Возвращает строковое представление объекта.

__repr__(self) - вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.

Вывод: приобрел навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.