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.

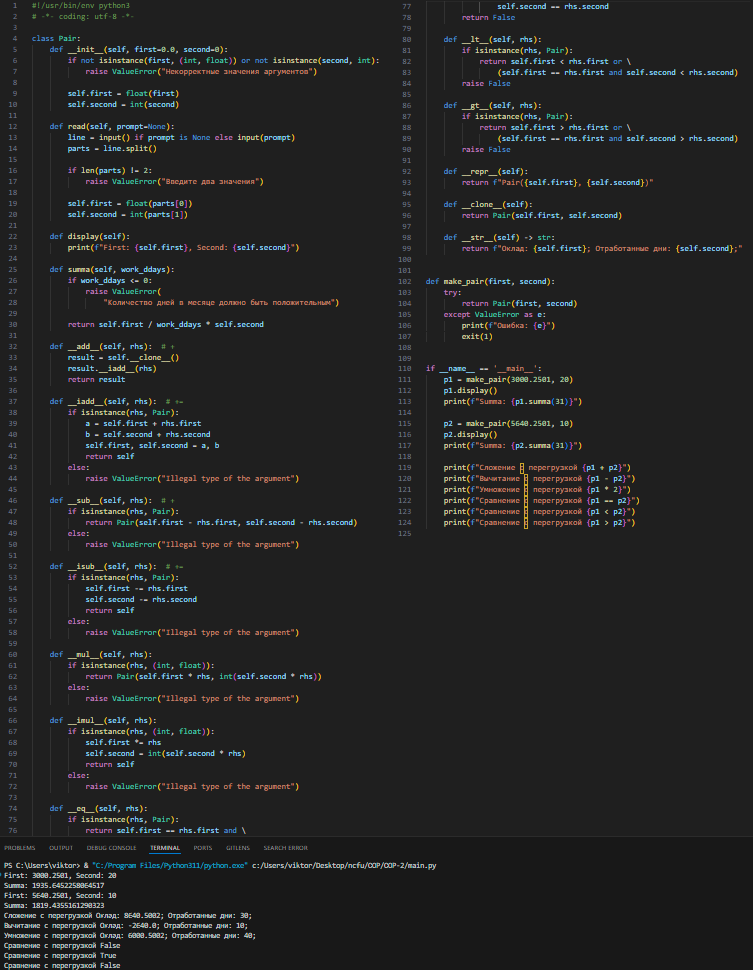


Рисунок 1. Код решения индивидуального задания 1 и его выполнение

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.

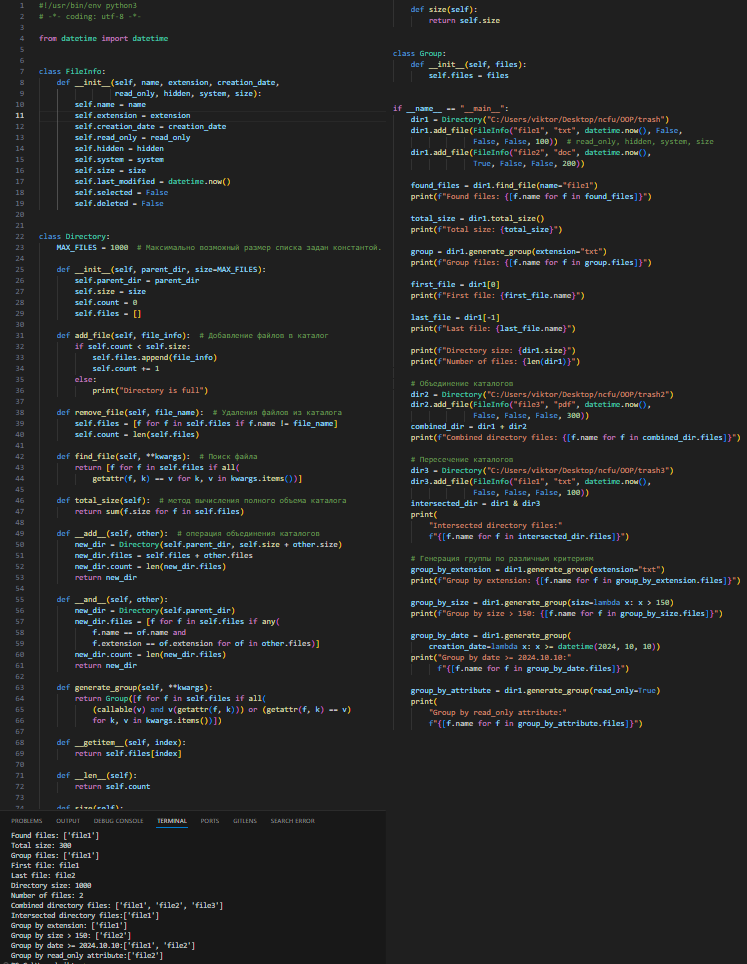


Рисунок 2. Код решения индивидуального задания 2 и его выполнение

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.